GOVINDA DASA COLLEGE, SURATHKAL
(AFFILIATED TO MANGALORE UNIVERSITY)
(2023-2024)

PROJECT REPORT
ON
"PARK BOOK"

Submitted to Mangalore University, in partial fulfilment of the requirements for the awards of degree of Bachelor of Computer Application.

PROJECT ASSOCIATES

DEVESH
(U05GD21S0008)

Under the valuable guidance of

Internal Guide

Mrs. Shailaja
Department Of Computer Science
Govinda Dasa College,
Surathkal.

GOVINDA DASA COLLEGE, SURATHKAL
(AFFILIATED TO MANGALORE UNIVERSITY)
(2023-2024)

CERTIFICATE

This is to certify that the project report entitled, "**PARK BOOK**" is an authenticated record of the work carried out by, **DEVESH** (U05GD21S0008) as partial fulfilment of the requirement for the award of Bachelor's Degree in Computer Application of Mangalore University has worked under by guidance and supervision during year 2023-2024.

Forward to Principal for Approval

(**PROJECT GUIDE**)   (**HOD**)

Approved and forward to Mangalore University

Place: Surathkal
Date:
Examiners: (**Principal**)
1.
2.

<u>DECLARATION</u>

We hereby declare that this project work "**PARK BOOK**", has been done by us under the guidance of Mrs. Shailaja H (Internal).

 We also declare that this report is the result of the due guidance of the Internal and our effort and this project has not been submitted to any other University.

Date:

Place: Surathkal

DEVESH

(U05GD21S0008)

Sign:

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any work would be incomplete without a mention of the people who made it possible, whose constant guidance and encouragement served as beacon light and crowned our effort with success.

We are honoured to extend our heartful gratitude to the Management of "Govinda Dasa College".

Sincere thanks to H.O.D Mrs. Geetha K, Department of BCA and our project guide Mrs. Shailaja H department of BCA, Govinda Dasa College, Surathkal Mangalore without whom this project has been impossible.

Thanks to Mangalore University for giving us an opportunity to take up this project as the knowledge gained by working on the project is very valuable.

We are also grateful to the principal of a college Prof. P. Krishnamoorty for allowing us to do the project in this institution.

This project is a team effort. The success of this project we believe is due to the team organization, coordination and understanding. The credit for the success of this project goes to the team as whole.

We would be failing in our duty if we do not thank our parents, lecturers, friends and almighty on which we all relied on, for help and encouragement throughout our BCA course and especially during our project work.

Thanking you all,

Yours faithfully,

DEVESH (U05GD21S0008)

CONTENTS

PARK BOOK

SYSNOPSIS

PARK BOOK

PARK BOOK

Govinda Dasa College, Surathkal

PARK BOOK

1. INTRODUCTION

The proposed project "PARK BOOK" is a smart parking booking system that provides customers an easy way of reserving a parking space online. It overcomes the problem of finding a parking space in commercial areas that unnecessary consumes time. Hence this project offers a web based reservation system where users can view various parking areas and select the space to view whether space is available or not. If the booking space is available then he/she can book it for specific time slot. The booked space will be marked red and will not be available for anyone else for the specified time.

1.1 Modules:

- **Admin Login:** The system is under supervision of admin who manages the bookings made.
- **User login/registration:** Users have to first register themselves to login into the system.
- **Parking availability check:** User can click on spaces to view the availability. If the space is already booked it will be marked yellow and the available ones will be seen in normal color.
- **Parking booking:** Users can book parking space for their required date and time.
- **Check-in and Check-out**: Check-in is the initial registration or entry procedure into a system or facility, requiring the provision of relevant details for access. Whereas check-out marks as the conclusion or exit phase of an interaction.
- **User verify**: This module validates user's phone numbers to ensure accurate communication and access to system features.
- **Slot add/remove:** The admin interface allows for seamless addition and removal of parking slots as needed.
- **Update price:** The admin interface enables easy updating of parking prices to reflect changes in demand or operational requirements.

1.2 User side functionality:

- Book parking space
- Check available spot

PARK BOOK

- Manage booking history

1.3 Admin side functionality:

- Add or remove slots
- Modify the prices
- Manage check-in and check-out function.

1.4 Advantages:

- Users can get learn about parking areas for particular locations.
- It saves user time in search of parking space available in such a long parking area.
- The system provides a graphical view of the parking spaces.
- User can confirm their spot online and confirm their space.
- It excludes the need of human efforts for managing parking spaces.
- The system generates online bill for requested time and even sends an email.
- Cost-effective.

1.5 Disadvantages:

- It requires an internet connection.
- Does not allow the customer to cancel the order once the product is ordered.

1.6 Applications:

- The project can be implemented in commercial areas for employee parking.
- It can be utilized by companies and organizations (hospitals, schools, colleges) to automate their parking system.
- The system can also be used in public places for public parking like in malls, station, and so on.

1.7 End User:

- Public user

PARK BOOK

1.8 Structure of the software system:

Govinda Dasa College, Surathkal
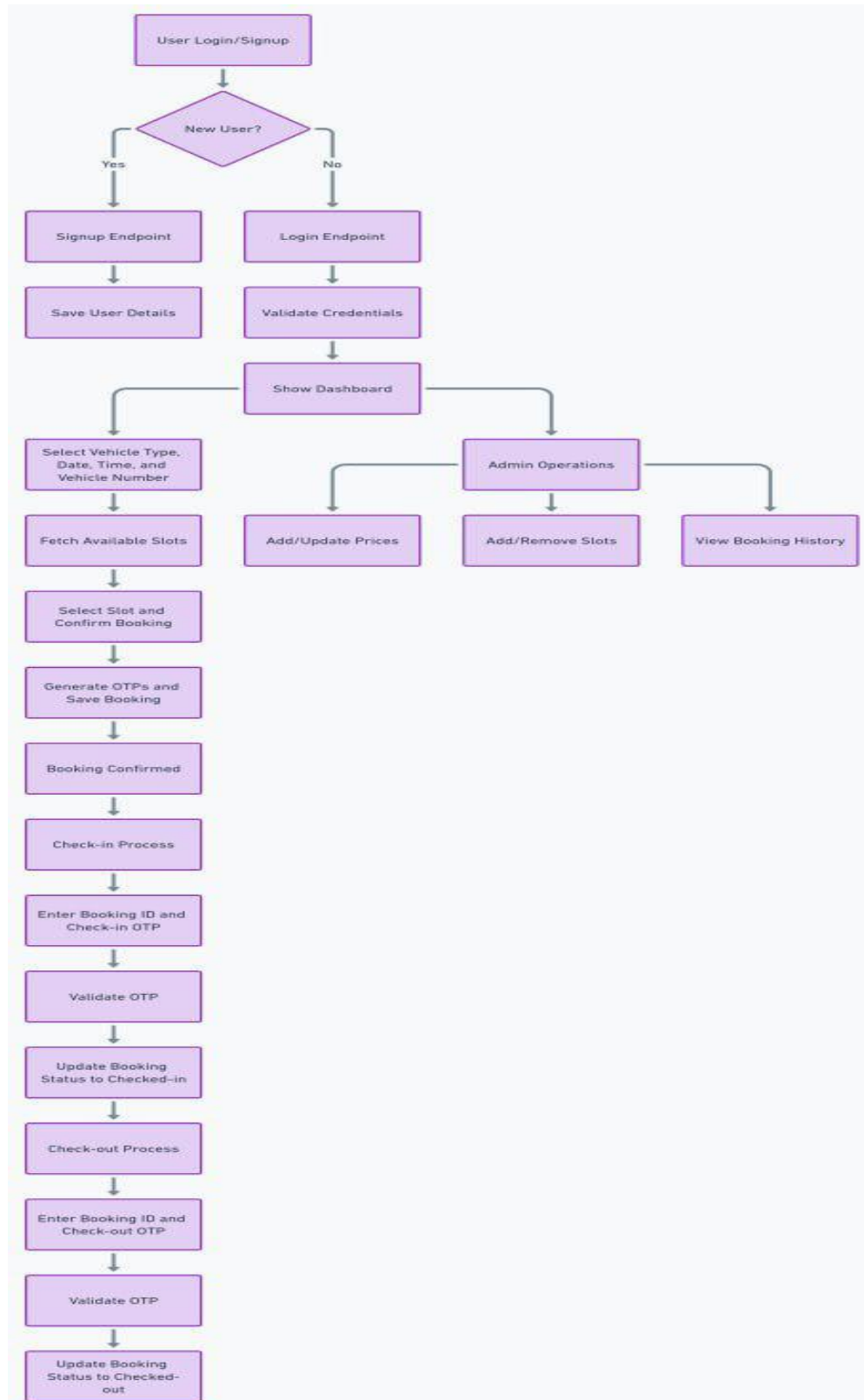
SOFTWARE REQUIREMENT SPECIFICATION (SRS)

PARK BOOK

2.SRS

2.1 Introduction(about SRS)

The Software Requirement Specification (SRS) is a document that outlines the detailed description of a software system. In the case of the "Park Book", the SRS serves as a comprehensive guide that describes the requirements and functionalities of the system. It acts as a reference for both the development team and stakeholders involved in the project.

2.2 Overall Description

2.2.1 Product Perspective

The Parking Booking System is a software application designed to streamline and simplify the process of reserving spaces. It provides a convenient solution for both parking lot owners and users, offering a range of features and benefits.

For parking lot owners, the system offers a centralized platform to manage and monitor their parking spaces. They can easily set up their parking lot details. The system allows them to track reservations, view occupancy rates, and generate reports for better analysis and decision-making. Additionally, the system provides a secure payment gateway to ensure seamless transactions with customers.

For users, the parking booing system offers a hassle-free way to find and reserve parking spaces. They can search for available parking lots based on their date, time. The system provides real-time availability updates, allowing users to make informed decisions. Users can also view parking lot details, to ensure they choose the best option for their needs. The system offers a simple booking process, allowing users to reserve and pay for their parking space with just a few clicks.

Overall, the parking booking system aims to improve the parking experience for both parking lot owners and users. It simplifies the process of finding and reserving parking spaces, reduces the need for manual coordination, and

PARK BOOK

enhances efficiency. With its user-friendly interacts and comprehensive features, the system provides a convenient solution for all parties involved.

2.2.2 Product Functions

Product functions are the specific capabilities and features that the Parking Booking System provides to users and administrators to facilitate parking space reservation and management.

- **User Registration and Authentication:** Users can create accounts and securely log in to access the system, ensuring personalized and secure access to booking features.
- **Parking Space Management:** Administrators can add, remove, and modify parking spaces within the system, ensuring accurate and up-to-date availability for users.
- **Booking Management:** Users can search for available parking spaces based on criteria such as location, date, and time, and book them seamlessly through the system.
- **Reservation Management:** Users have the ability to view, modify, or cancel their parking reservations as needed, providing flexibility and convenience.
- **Admin Panel:** An intuitive interface is provided for administrators to manage users, parking spaces, reservations, and system settings efficiently and effectively.
- **Mobile Accessibility:** The system is designed to be accessible via mobile devices, allowing users to book parking spaces conveniently while on the go, enhancing user experience and accessibility.

2.2.3 User Characteristics

- **Drivers:** These users are the primary target audience of the Parking Booking System. They are individuals who own or operate vehicles and require parking spaces for various durations, such as commuters, shoppers, or event attendees.

- **Administrators:** These users are responsible for managing the Parking Booking System. They include parking lot owners, facility managers, or system administrators who oversee the administration, configuration, and maintenance of the system.

PARK BOOK

- **Parking Lot Owners/Operators:** Another important group of users for a parking booking system are the parking lot owners and operators. These individuals or organizations provide the parking spaces that are made available for reservation.

2.2.4 General Constraints

- **Technical Constraints:**
    - Compatibility: The system must be compatible with various web browsers and mobile devices to ensure accessibility for users.
    - Scalability: The system should be scalable to accommodate increasing numbers of users and parking spaces over time.
    - Security: The system must adhere to industry-standard security practices to protect user data and payment information.

- **Regulatory Constraints:**
    - Compliance: The system must comply with local regulations and ordinances related to parking, data privacy, and online payments.
    - Accessibility: The system should adhere to accessibility standards to ensure equal access for individuals with disabilities.

- **Budgetary Constraints:**
    - Cost: Development and maintenance costs must align with the allocated budget for the project.
    - Resource Constraints: Limited resources, such as time and manpower, may impact the scope and timeline of the project.

- **Operational Constraints:**
    - Availability: The system should have minimal downtime to ensure uninterrupted access for users.
    - Performance: The system must perform efficiently, with fast response times and minimal latency, even during peak usage periods.
    - Reliability: The system should be reliable, with robust backup and recovery mechanisms to prevent data loss and ensure continuity of service.

- **Environmental Constraints:**

- Infrastructure: The availability of physical infrastructure, such as parking lots equipped with necessary technology, may influence system implementation.
- Geographic Constraints: The system may need to consider geographic factors such as location-specific parking regulations and availability.

- **Integration Constraints:**
  - Integration with Existing Systems: The system may need to integrate with existing parking management systems, payment gateways, or third-party services.

2.2.5 Assumptions

- Users are expected to have reliable internet connectivity to access the system for booking parking spaces.
- The payment processing system is assumed to function smoothly, provided users input accurate payment information.
- Users rely on the accuracy of parking space availability information provided by the system, with occasional discrepancies possible due to real-time factors.
- Users are responsible for adhering to terms of service, rules, and regulations set by the Parking Booking System and associated parking facilities.
- The Parking Booking System is assumed to prioritize security measures to protect user data and payment information.

2.3 Special Requirements

2.3.1 Software Requirements

- Web and Mobile Application.
- Browsers such as Internet Explorer, Google Chrome, Mozilla Firefox etc.
- Database Management System (Mongo DB).

2.3.2 Hardware Requirements

PARK BOOK

- Server and Data Storage.
- Mobile Device Connectivity.
- Network Infrastructure.

2.3.3 Communication Interface

- The project must use the HTTP protocol for communication over the internet and for the internet communication will be through TCP/IP protocol suite.
- The user must connect to the internet to access the website.

2.4 Functional Requirements

2.4.1 ADMIN

An administrator (admin) is a user with privileged access rights responsible for managing and overseeing the operation of the Parking Booking System.

- **Parking Space Management:** Admins add, modify, and remove parking spaces.
- **Reservation Management:** Admins view, modify.
- **User Management:** Admins view, modify, suspend, or delete user accounts.
- **Reporting and Analytics:** Admins generate reports on system usage, revenue, etc.
- **System Settings:** Admins configure system settings such as pricing and notifications.

2.4.2 USER

A user is an individual who interacts with the Parking Booking System to search for and reserve parking spaces.

- **User Registration and Authentication:** Users create accounts and log in securely to access the system.
- **Parking Space Search and Booking:** Users search for available parking spaces based on location, date, and time. Users select and book parking spaces for specific dates and times.

PARK BOOK

- **Reservation Management:** Users view, modify, or cancel their parking reservations as needed.
- **Mobile Accessibility:** Users access the system conveniently via mobile devices for on-the-go booking.

2.5 Design constraints:

Design constraints refer to limitations or restrictions imposed on the design of a system, often stemming from technical, regulatory, or practical considerations. These constraints influence design decisions and shape the overall architecture and functionality of the system.

- **User Interface Consistency:** Ensure uniformity in the design elements such as buttons, fonts, and colors throughout the system to provide a cohesive user experience.
- **Responsive Design:** Design the system to adapt seamlessly to various screen sizes and resolutions, ensuring optimal user experience across devices.
- **Cross-Browser Compatibility:** Ensure that the system functions correctly and displays consistently across different web browsers to accommodate users' preferences.
- **Accessibility Compliance:** Design the system in accordance with accessibility standards to ensure it is usable by individuals with disabilities, including features such as screen reader compatibility and keyboard navigation.
- **Performance Optimization:** Optimize the system's performance to minimize loading times and provide a smooth user experience, even under high traffic conditions.
- **Scalability:** Design the system architecture to accommodate growth in user base and parking space inventory without sacrificing performance or stability.
- **Data Security:** Implement robust security measures to safeguard user data, transactions, and system integrity from unauthorized access, breaches, or data loss.

2.6 System attributes:

PARK BOOK

System attributes are the characteristics or qualities that define the behaviour, performance, and overall quality of the Parking Booking System. These attributes guide the design, development, and evaluation of the system to ensure it meets the needs and expectations of users and stakeholders.

- **Reliability:** The system's ability to perform consistently and accurately, ensuring users can rely on it for booking parking spaces without unexpected failures or errors.
- **Availability:** The system's ability to be accessible and operational when needed, ensuring users can access and use it whenever they require parking reservations.
- **Scalability:** The system's ability to accommodate increasing user demand and parking space inventory over time, ensuring it can scale effectively without sacrificing performance or stability.
- **Performance:** The system's responsiveness and efficiency in processing user requests, search queries, and payment transactions, providing a seamless and fast user experience.
- **Security:** The system's measures to protect user data, payment information, and system integrity from unauthorized access, breaches, or vulnerabilities, ensuring confidentiality, integrity, and availability.
- **Usability:** The system's ease of use, intuitive interface, and user-friendly design, ensuring users can navigate, search for parking spaces, and make reservations with minimal effort and training.
- **Maintainability:** The system's ease of maintenance, updates, and troubleshooting, ensuring it can be efficiently managed and enhanced over time to address evolving user needs and technology changes.
- **Flexibility:** The system's ability to adapt to changing requirements, regulations, or technological advancements, ensuring it remains relevant and effective in meeting user needs in dynamic environments.

12

SYSTEM

DESIGN

3. SYSTEM DESIGN

3.1 Introduction (about System Design)

System design refers to the process of defining the architecture, components, modules, interfaces, and data flows of a software system to meet specified requirements and objectives. It involves translating the system requirements into a detailed blueprint that guides the implementation, testing, and deployment phases of the software development lifecycle. The system design encompasses both the high-level architectural design and the detailed design of individual components, ensuring that the system is scalable, reliable, maintainable, and meets the needs of its users and stakeholders.

3.2 Assumption and Constraints

Assumptions:
1. Users have reliable internet access.
2. Users provide accurate payment details.
3. Parking space availability information is generally accurate.
4. Users adhere to system regulations.
5. Security measures protect user data effectively.

Constraints:
1. The system's UI must adhere to accessibility standards.
2. Scalability is essential to handle increased demand.
3. Compliance with data privacy regulations is mandatory.
4. Integration with external services may impose technical constraints.
5. Limited resources such as time, budget, and manpower may impact project scope.

3.3 Functional Decomposition

3.3.1 System Software Architecture

Software architecture is simply, the organization of system. The organization includes all the components, how they interact with each other, the environment in which they operate and the principle used to

PARK BOOK

design the software. In many cases, it can also include the evolution of the software into the future.
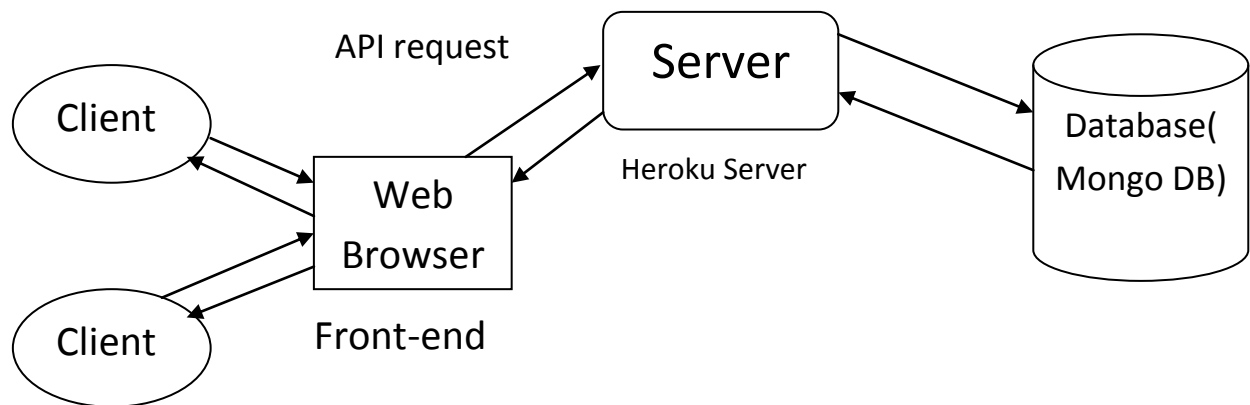
3.3.2 System Technical Architecture

Technical architecture (TA) is a form of IT architecture that is used to design the   computer system. It involves the development of technical blueprint with regard to the arrangement interaction, interdependence of all elements so that system-relevant requirement is met.

PARK BOOK
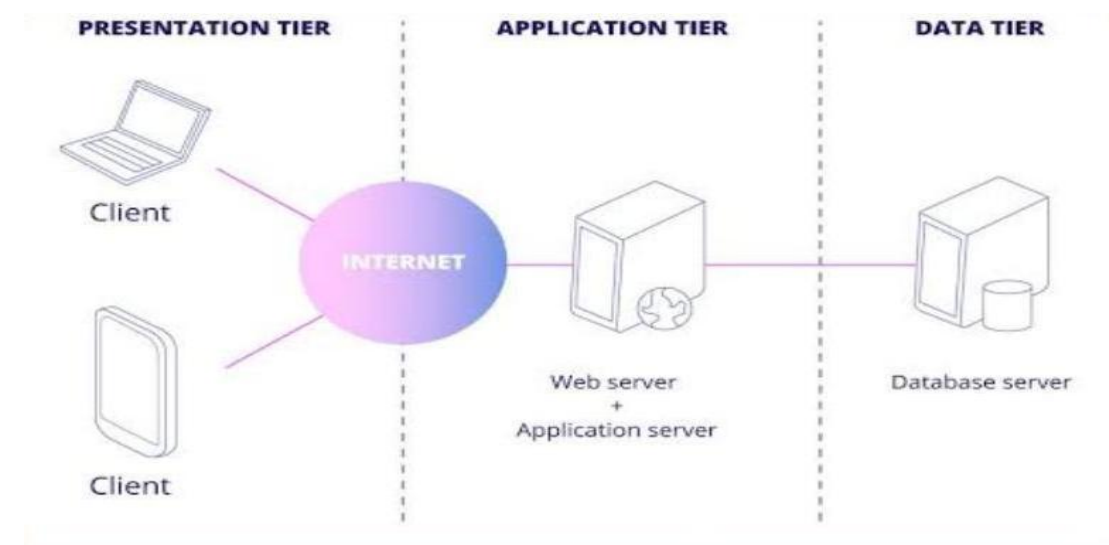
3.4 Description of Programs

3.4.1 Context Flow Diagram (CFD)

A Context Flow Diagram (CFD), also known as a Level 0 DFD (Data Flow Diagram), provides a high-level overview of the entire system, showing the interactions between the system and external entities. In the context of the Parking Booking System, the CFD illustrates how users interact with the system to book parking spaces and how the system interacts with external entities such as payment gateways and administrative interfaces.

```
┌──────────┐      ┌──────────┐      ┌──────────┐
│  ADMIN   │ ←──→ │ ParkBook │ ←──→ │ CUSTOMER │
└──────────┘      └────┬─────┘      └──────────┘
                       ↕
                  ┌─────────┐
                  │DATABASE │
                  └─────────┘
```
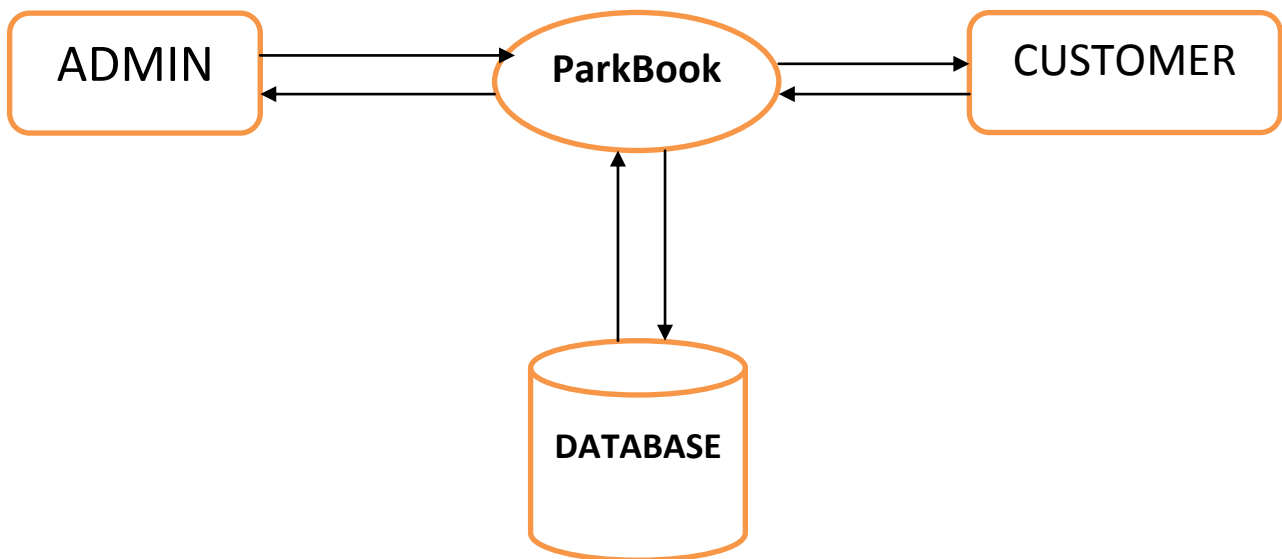
3.4.2 Data Flow Diagram (DFD)

A Data Flow Diagram is a graph showing the flow of data values from their sources in through processes that transform them to their destination in other object.

A DFD also known as "bubble chart", has the purpose of clarifying the system require and identifying major transformations that will become programs system design. So, it is for starting to the design phase that functionally decomposes the requirements specifications down to the lowest level of detail. A DFD consists of series of a bubbles joined by lines. The bubbles represent data transformations and lines represent data flows in system.

3.4.2.1 DFD Symbols

Name	Notation	Description
Process		Represents a function or process that transforms input data. It is denoted by a rectangle with rounded corners.
External entity		Represents entities outside the system boundary that interact with the system. It is denoted by a square.
Data flow	⟶	Represents the flow of data between processes, data stores and external entities. It is denoted by an arrow.
Data source		Data stores are repositories of data in the system. They are sometimes also referred to as files.

3.5 Description of Components

3.5.1 Functional Component 1: ADMIN

 - **Dashboard:** Provides administrators with an overview of system activities, including the number of bookings, revenue generated, and system health indicators.

 - **Manage Parking Spaces:** Allows administrators to add, modify, or remove parking spaces from the system, including details such as location, capacity, and availability.

PARK BOOK

 - View Reservation Logs: Enables administrators to view logs of all parking reservations, including booking details, user information, and payment status.

 - Manage User Accounts: Allows administrators to manage user accounts, including creating new accounts, updating user information, and suspending or deleting accounts as needed.
 - System Settings: Provides administrators with access to system settings, allowing them to configure parameters such as pricing, notification preferences, and integration with external services.

3.5.2 Functional Component 2: USER

- Search for Parking Spaces: Allows users to search for available parking spaces based on location, date, and time preferences.
   - Book Parking Space: Enables users to select and reserve a parking space for a specific date and time duration.
   - View Reservation Details: Allows users to view details of their existing parking reservations, including booking dates, times, and parking space information.
- Check-in and Check-out: Check-in is the initial registration or entry procedure into a system or facility, requiring the provision of relevant details for access. Whereas check-out marks as the conclusion or exit phase of an interaction.
  • **User verify**: This module validates user's phone numbers to ensure accurate communication and access to system features.

DATABASE

DESIGN

4. DATABASE DESIGN

4.1 Introduction (about Database Design)

The database design plays a crucial role in storing, managing, and retrieving data related to parking spaces, reservations, users, and system configurations. A well-designed database ensures data integrity, efficiency, and scalability, facilitating seamless operation and optimal performance of the system.

4.2 Purpose and Scope

Purpose:

The purpose of the Parking Booking System is to provide a convenient and efficient platform for users to search for, book, and manage parking spaces. It aims to streamline the process of parking reservation, reduce parking congestion, and improve overall user experience by offering a user-friendly interface, real-time availability updates, and secure payment processing. The system also serves to optimize parking space utilization, increase revenue for parking facility owners, and enhance administrative efficiency through centralized reservation management and reporting capabilities.

Scope:

1. User Interface: Providing an intuitive and user-friendly interface for users to search for available parking spaces, make reservations, and manage bookings.
2. Reservation Management: Allowing users to view, modify, and cancel parking reservations, as well as receive automated notifications regarding booking status.
3. Administrative Interface: Providing administrators with tools to manage parking spaces, reservations, users, and system settings, as well as access reporting and analytics features.
4. Data Management: Storing and managing data related to parking spaces, reservations, users, and system configurations in a centralized database, ensuring data integrity and security.
5. Integration: Integrating with external services such as mapping APIs, traffic data providers, and payment gateways to enhance system functionality and user experience.

PARK BOOK

6. Security: Implementing robust security measures to protect user data, payment information, and system integrity from unauthorized access, breaches, and vulnerabilities.

4.3 Table Definition

Table Name: USER

Fields	Data type
userId	String
email	String
password	String
isAdmin	Boolean
phoneNo	String
isVerified	Boolean

Table Name: BOOKING

Fields	Data type
bookingId	String
userId	String
slotId	String
vehicleType	String
vehicleNumber	String
bookedFrom	DateTime
bookedTill	DateTime
amount	Integer
isCheckedIn	Boolean

Govinda Dasa College, Surathkal

isCheckedOut	Boolean
checkinOtp	String
checkoutOtp	String
orderTime	DateTime

Table Name: SLOT

Fields	Data type
slotId	String
type	String
isOccupied	Boolean

Table Name: PRICE

Fields	Data type
priceId	String
carPrice	Integer
bikePrice	Integer
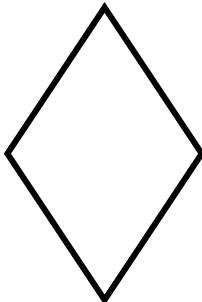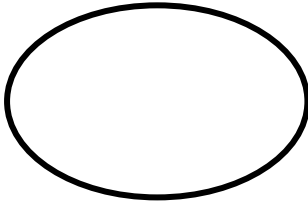
4.4 ER Diagram

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.
Entity relationship analysis uses three major abstraction to describe data. These are entities which are distinct things in the enterprise. Relationships are meaningful interaction when the objects and the attributes which are the
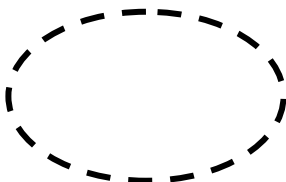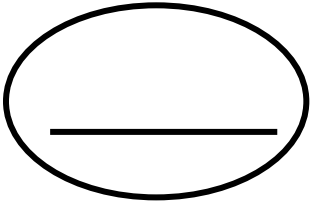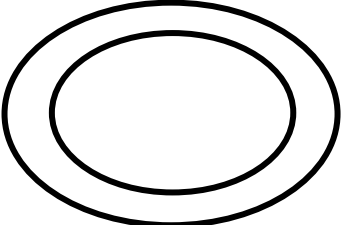
PARK BOOK

properties of entities and relationship. We can express the overall logical structure of the database graphically with an E-R diagram. The Symbols are shown below the table.

Name	Notation	Description
Entity		The entity is represented by a box within the ERD. Entities are abstract concepts, each representing one or more instances of the concept in question. An entity might be considered a container that holds all of the instances of a particular thing in a system.
Relationship		Relationships are represented by Diamonds. A relationship is a named collection or association between entities or used to relate to two or more entities with some common attributes or meaningful interaction between the objects.
Link		Lines link attributes to entity sets or entity sets to relationship sets.
Attributes		Attributes are represented by Oval. An attribute is a single data item related to a database object. The database scheme associates with each database entity.

Derived Attribute		Dashed ellipse denotes derived attributes.
Key Attribute or Single Valued Attribute		As entity type usually has an attribute whose values are distinct for each individual entry in the entity set. It is represented by an underline word in ellipse.
Multivalued Attribute		Attributes that have different numbers of values for a particular attribute. It is represented by a double ellipse.
Cardinality Ratio	**1:1** **1:N** **N:1** **N:N**	It specifies the maximum number of relationships instances that an entity can participate in. There are four cardinality ratios.

23

PARK BOOK

Govinda Dasa College, Surathkal
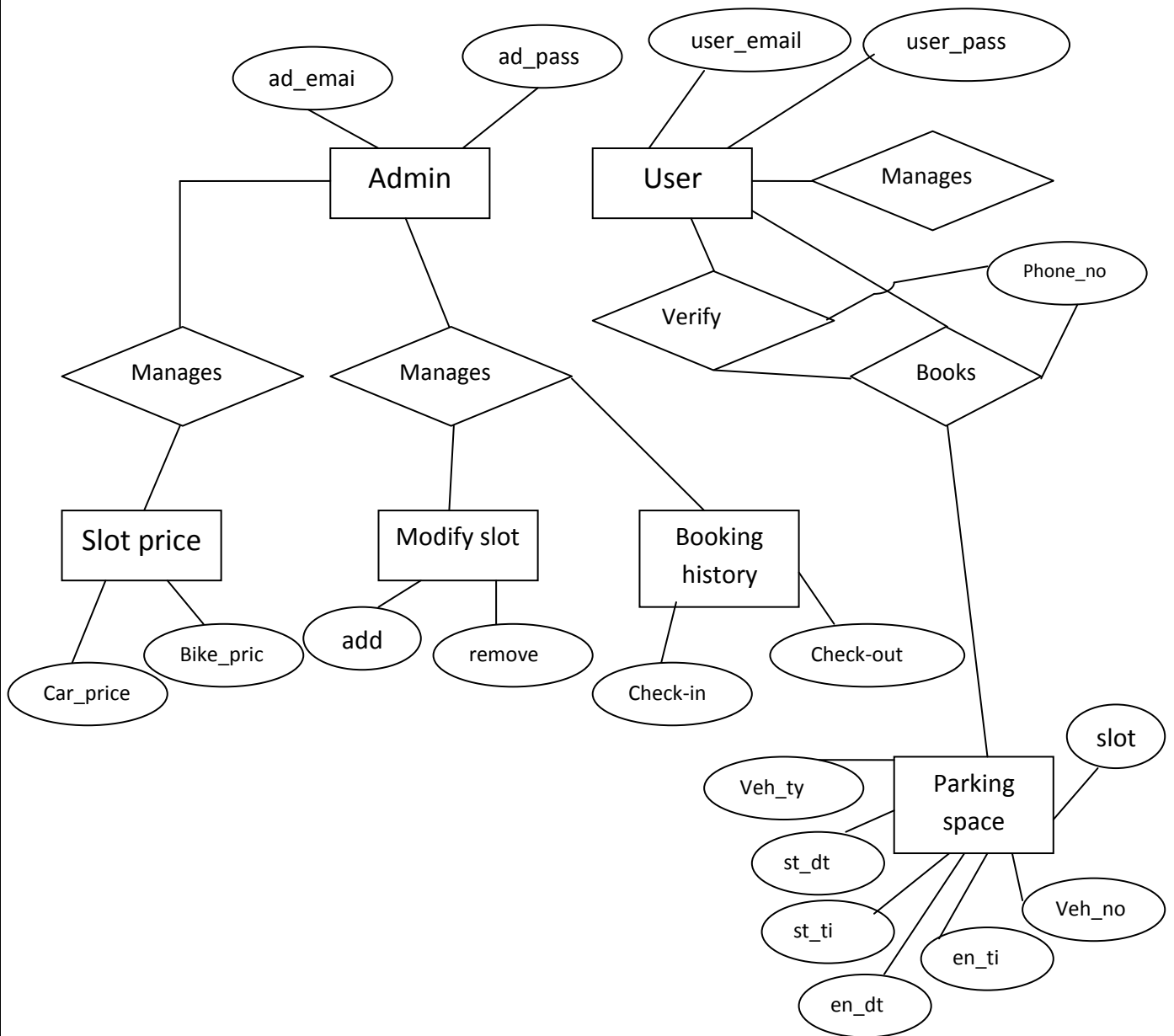
DETAILED

DESIGN

5. DETAILED DESIGN

5.1 Introduction (about Detailed Design)

The purpose of the design phase is to plan a solution of the problem specified by the requirements document. This phase is the first step in moving from the problem domain to the solution domain. In other words, starting with what is needed; design takes us towards how to satisfy the needs. The design of the system is perhaps the most critical factor affecting the quality of the software; it has a major impact on the later phases, particularly testing and maintenance.

The design activity often results in three separate outputs- Architecture Design, High Level Design and Detailed Design. Architecture focuses on looking at a system as a combination of many different components, and how they interact with each other to produce the desired results. The High Level Design identifies the modules that should be built for developing the system and the specifications of these modules. At the end of system design all the major data structures, file formats, output formats etc., are also fixed. In detailed design, the internal logic of each of the modules are specified.

The design process for the software systems often has two levels. At the first level, the focus is on deciding which modules are needed for the system, the specifications of these modules and how the modules should be interconnected. This is what is called the System Design or the Top Level Design. In the second level, the internal design of the modules, or how the specifications of the module can be satisfied is decided. This design level is often called Detailed Design. Detailed design essentially expands the system design to contain a more detailed description of the processing logic and data structures so that the design is sufficiently complete for coding.

The Detailed Design refines the System Design hence the first applicable document here is system design document. We also refer the data structures. Hence the second applicable document is database design.

5.2 Structure of the software package

Various functional components used are:
- Admin module

- Users module

5.2.1 Module of Admin
This module is designed specifically for system administrators to manage and oversee various aspects of the system. It provides administrators with access to administrative functionalities such as managing parking spaces, update parking prices, admin history, reservation, users etc through a centralized interfaces.

5.2.1.1 Structure chart for Admin

```
           ┌──────────┐                    ┌──────────┐
           │  Admin   │────────────────────│  Auth0   │
           └──────────┘                    └──────────┘
                │                                │
           ┌──────────┐                  ┌───────────────┐
           │  Login   │                  │ Manage Users  │
           └──────────┘                  └───────────────┘
                │
   ┌────────────┼───────────────┬────────────────┐
┌──────────┐ ┌──────────┐ ┌──────────────┐ ┌──────────────────┐
│Add/Update│ │Add/Remove│ │ View Booking │ │     Manage       │
│  Prices  │ │  Slots   │ │   History    │ │ Checkin/Checkout │
└──────────┘ └──────────┘ └──────────────┘ └──────────────────┘
```
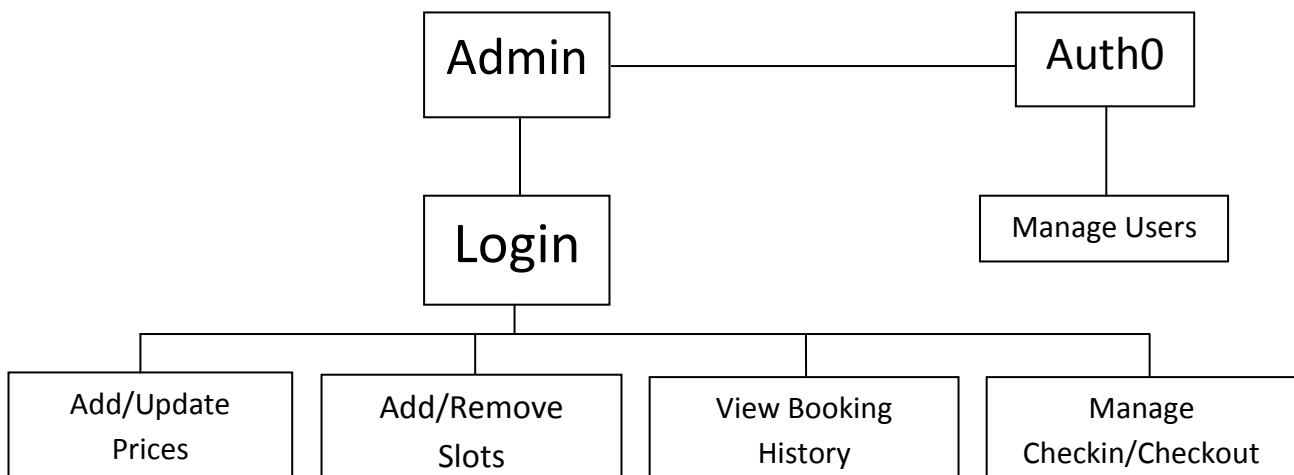
5.2.2 Module of Users
This module is designed to facilitate interactions between users and the system. It provides the functionalities for users can input search criteria, view available parking spaces, select preferred parking spaces, make reservations, manage bookings etc.

5.2.2.1 Structure chart for Admin

Govinda Dasa College, Surathkal

PARK BOOK

```
                    ┌──────────────┐
                    │    Users     │
                    └──────┬───────┘
                           │
                    ┌──────┴───────┐
                    │   Register   │
                    └──────┬───────┘
                           │
                    ┌──────┴───────┐
                    │    Login     │
                    └──────┬───────┘
```

| User verify | Input necessary details for booking | Fetch available slots | Select slot and confirm booking | Check-in Process | Check-out Process |
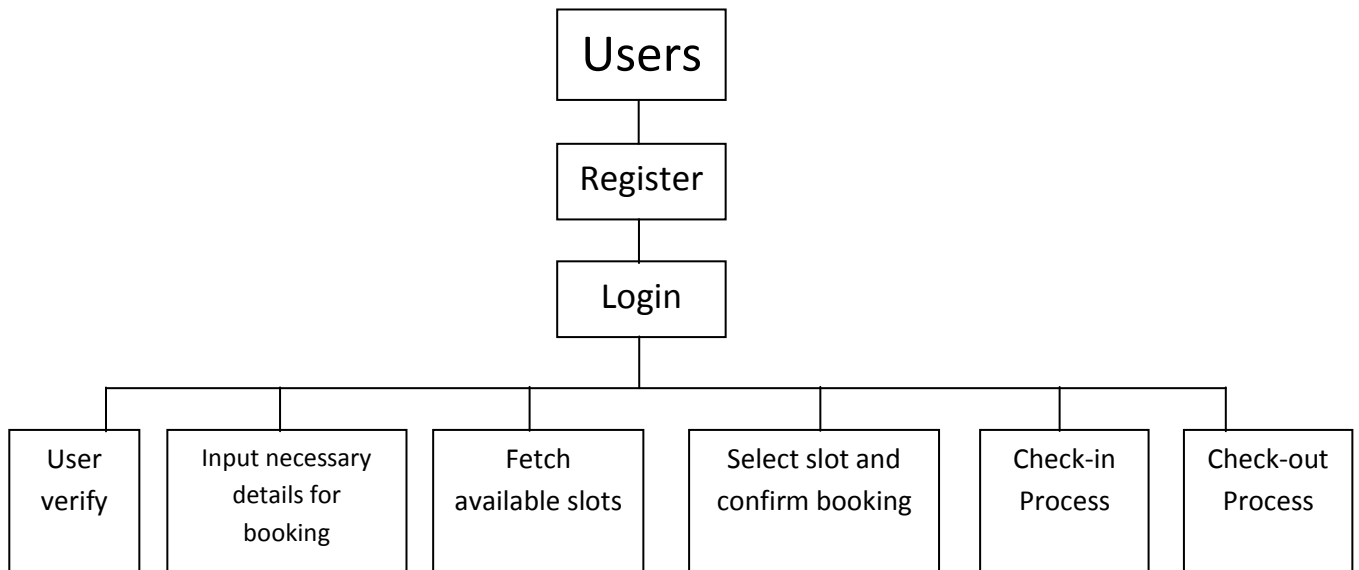
5.3 Modular Decomposition of the System

5.3.1 Admin
Login:

Input: ad_email, ad_pass;

Procedural Details:

```
                    ┌────────────┐
                    │   Start    │
                    └─────┬──────┘
                          │
                ┌─────────┴──────────┐
                │  Enter email ID    │◄────────┐
                │  and Password      │         │
                └─────────┬──────────┘         │
                          │                    │
                      ◇ Verified ?  ──False──► ┌──────────────┐
                          │                    │ Invalid Input│
                         True                  └──────────────┘
                          │
                ┌─────────┴──────────┐
                │     Display        │
                │    Dashboard       │
                └─────────┬──────────┘
                          │
                    ┌─────┴──────┐
                    │    Stop    │
                    └────────────┘
```

28

Govinda Dasa College, Surathkal
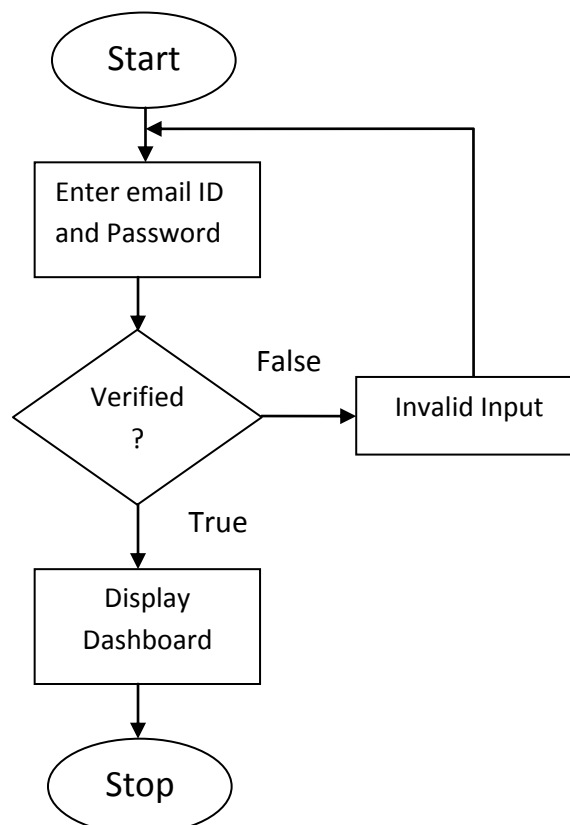
PARK BOOK

File I/O Interface: Admin page

Output: Email id and password will be checked for validity. If it is valid admin will be directed to admin page.

Add/Update prices :

Input: car_price, bike_price;

Procedural Details:
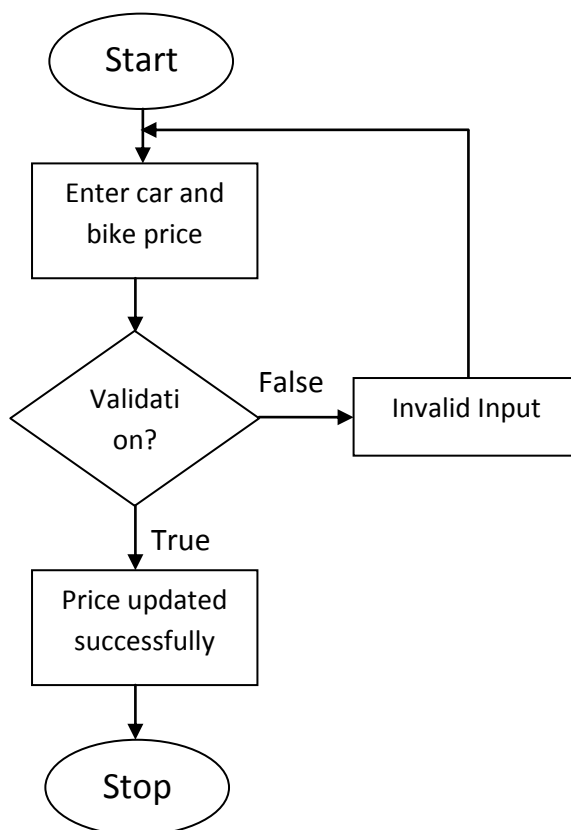
File I/O Interface: Price changing page.

Output: Admin can update vehicle prices.

Add/Remove slots :

Input: slot_id;

PARK BOOK

Procedural Details:

```
                          ( Start )
                             |
          +------------------+------------------+
          |                                     |
     [ Add Slot ]                        [ Remove Slot ]
          |                                     |
   / Enter slot id /                     / Enter slot id /
          |                                     |
     < Validation? > --False--> [ Invalid Input ]   < Validation? > --False--> [ Invalid Input ]
          | True                                   | True
   [ Slot added                              [ Slot removed
     successfully ]                            successfully ]
          |                                         |
          +------------------( Stop )---------------+
```
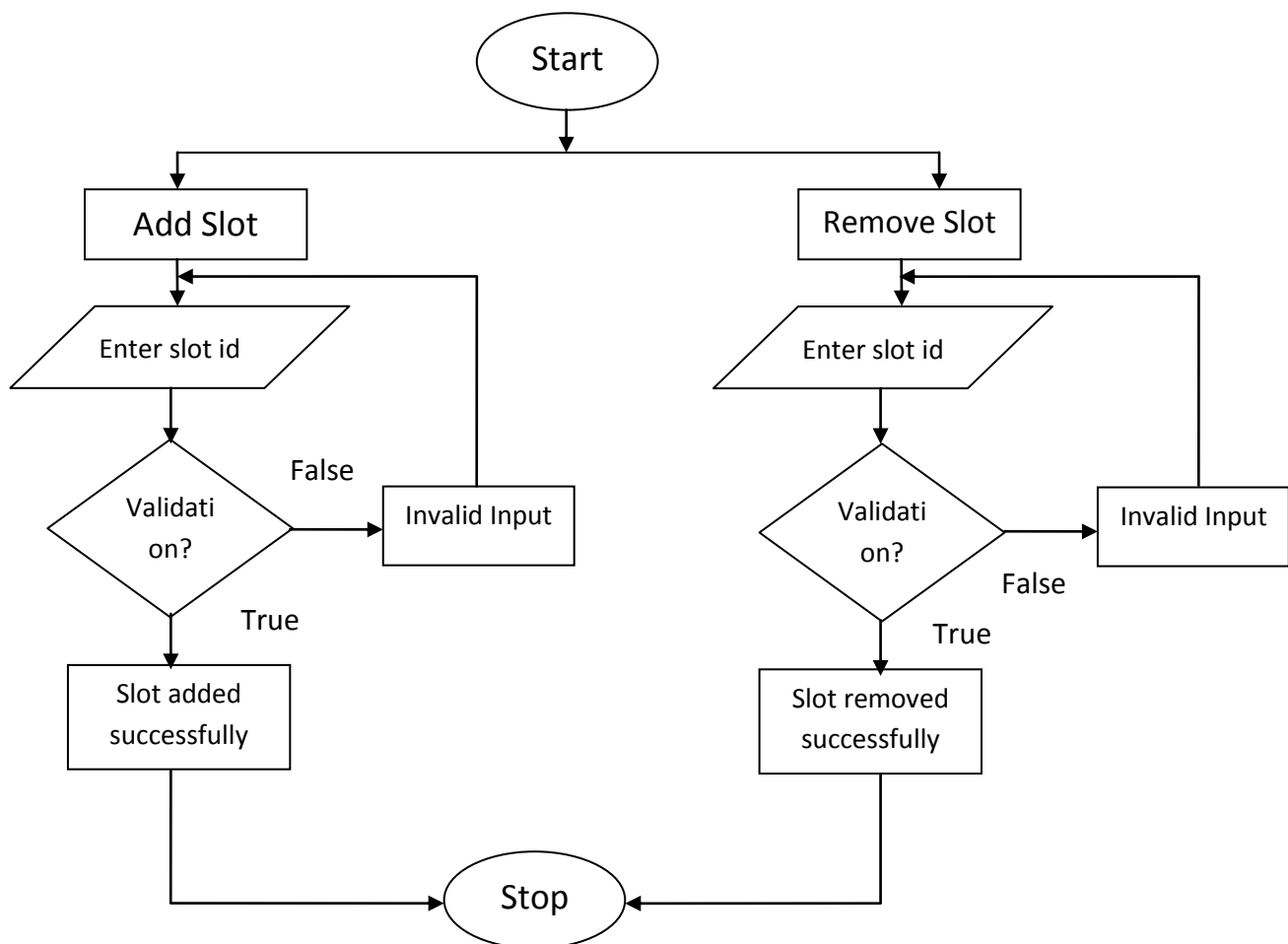
File I/O Interface: Slot add/remove page.

Output: Admin can add/remove slots.

View Booking History :

Input: vehicle_no, email;

Procedural Details:

PARK BOOK

```
                    ┌─────────┐
                    │  Start  │
                    └─────────┘
                         │
                         ▼
              ┌──────────────────┐
              │  Enter vehicle   │
              │  no. or email ID │
              └──────────────────┘
                         │
                         ▼          False      ┌──────────┐
                    ╱─────────╲ ──────────────▶ │No Record │
                   ╱ Validati  ╲               │  Found   │
                   ╲   on?     ╱               └──────────┘
                    ╲─────────╱
                         │ True
                         ▼
              ┌──────────────────┐
              │  Display Record  │
              └──────────────────┘
                         │
                         ▼
                    ┌─────────┐
                    │  Stop   │
                    └─────────┘
```
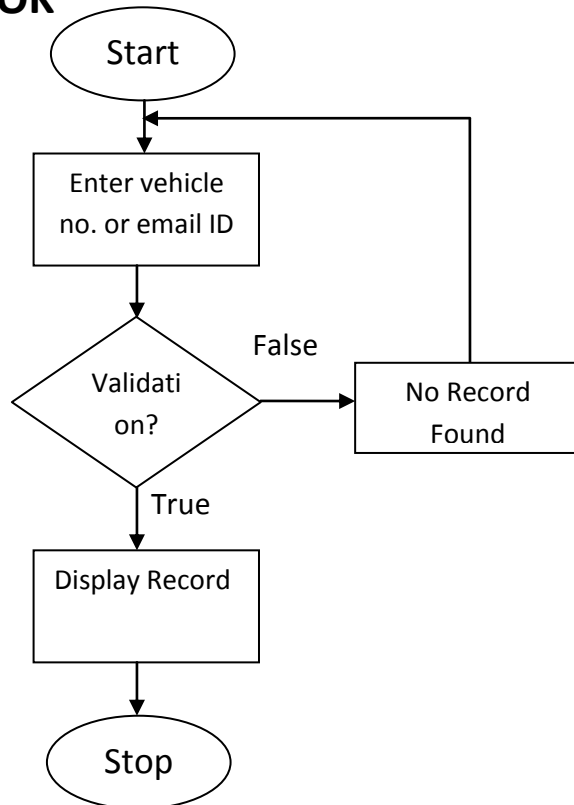
File I/O Interface: Admin history page.

Output: Admin can manage, view booking details.

Manage Users:

Input: user_id;

Procedural Details:

```
                    ┌─────────┐
                    │  Start  │
                    └─────────┘
                         │
                         ▼
              ┌──────────────────┐
              │ View public users│
              └──────────────────┘
                         │
                         ▼            False    ┌──────────────────────┐
                  ╱─────────────╲ ───────────▶ │No public users details│
                 ╱ If public users╲            │     available.        │
                 ╲  available?    ╱            └──────────────────────┘
                  ╲─────────────╱
                         │ True
                         ▼
              ┌──────────────────┐
              │Display public users│
              │    details.      │
              └──────────────────┘
                         │
                         ▼
                    ┌─────────┐
                    │  Stop   │
                    └─────────┘
```

Govinda Dasa College, Surathkal
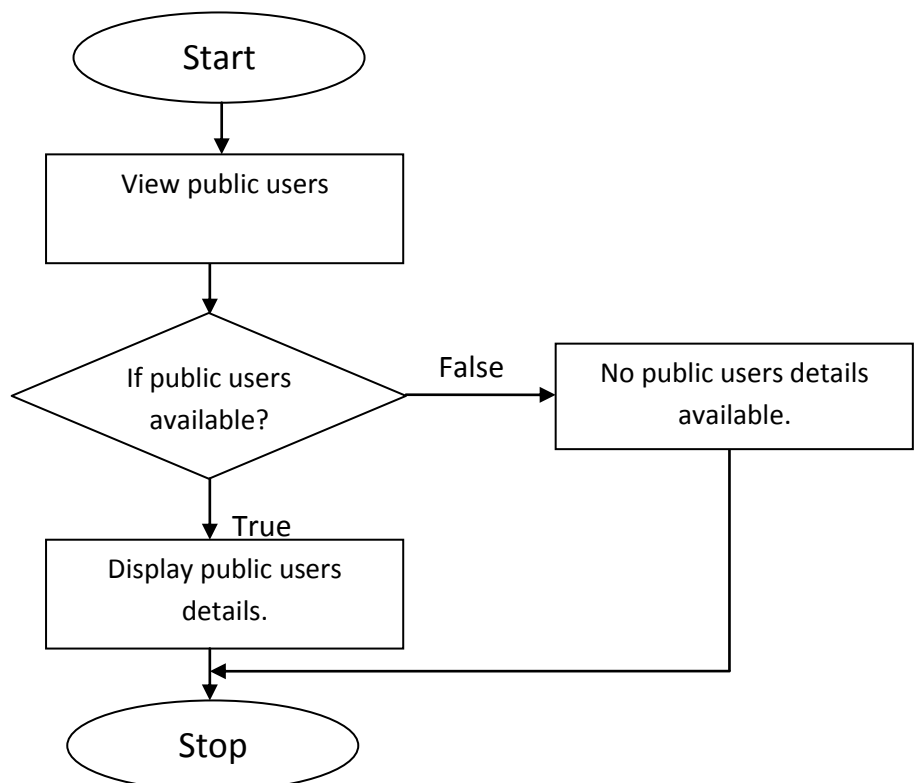
PARK BOOK

File I/O Interface: User management page.

Output: Admin can manage, add, delete users.

Manage Checkin/Checkout:

Input: otp;

Procedural Details:

```
                    ( Start )
                       |
                       v
              +------------------+
              |  Enter check-in  |
              |       OTP        |
              +------------------+
                       |
                       v
                   / Validati \  False    +-------------+
                  <    on?      >--------> | Invalid OTP |
                   \          /            +-------------+
                       |
                    True
                       v
              +------------------+
              |   Checked-in     |
              |  Successfully    |
              +------------------+
                       |
                       v
              +------------------+
              |  Enter check-    |
              |    out OTP       |
              +------------------+
                       |
                       v
                   / Validati \  False    +-------------+
                  <    on?      >--------> | Invalid OTP |
                   \          /            +-------------+
                       |
                    True
                       v
              +------------------+
              |   Checked-out    |
              |  Successfully    |
              +------------------+
                       |
                       v
                    ( Stop )
```
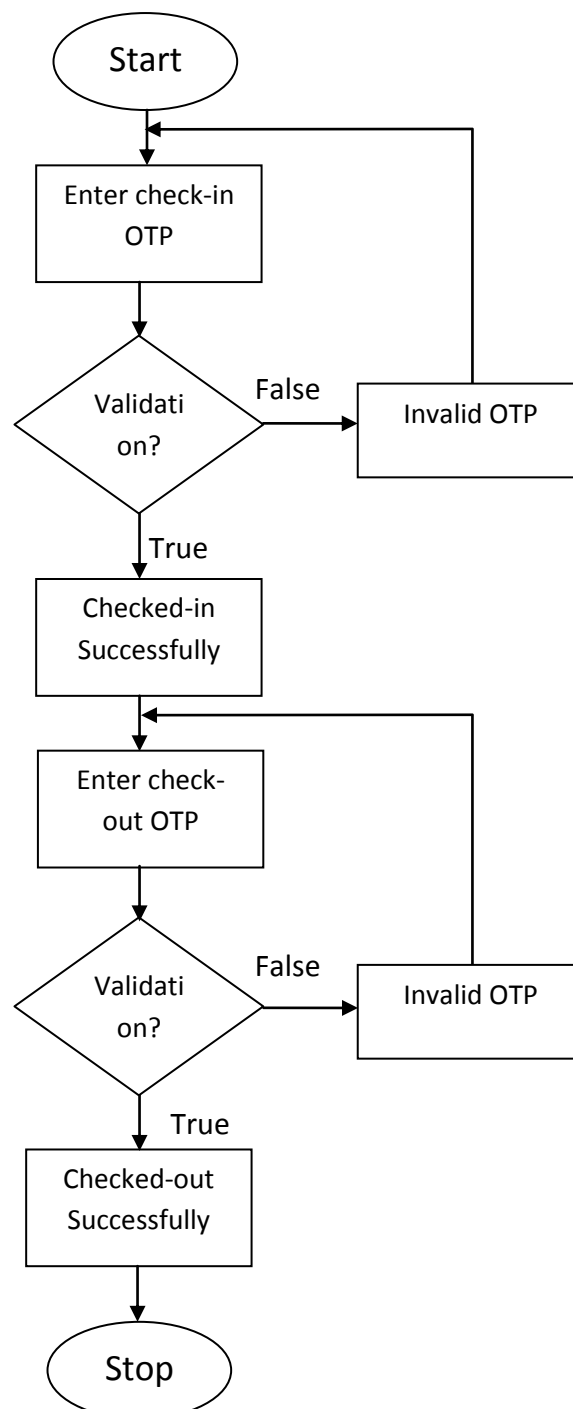
Govinda Dasa College, Surathkal

PARK BOOK

File I/O Interface: Check-in/Check-out table.

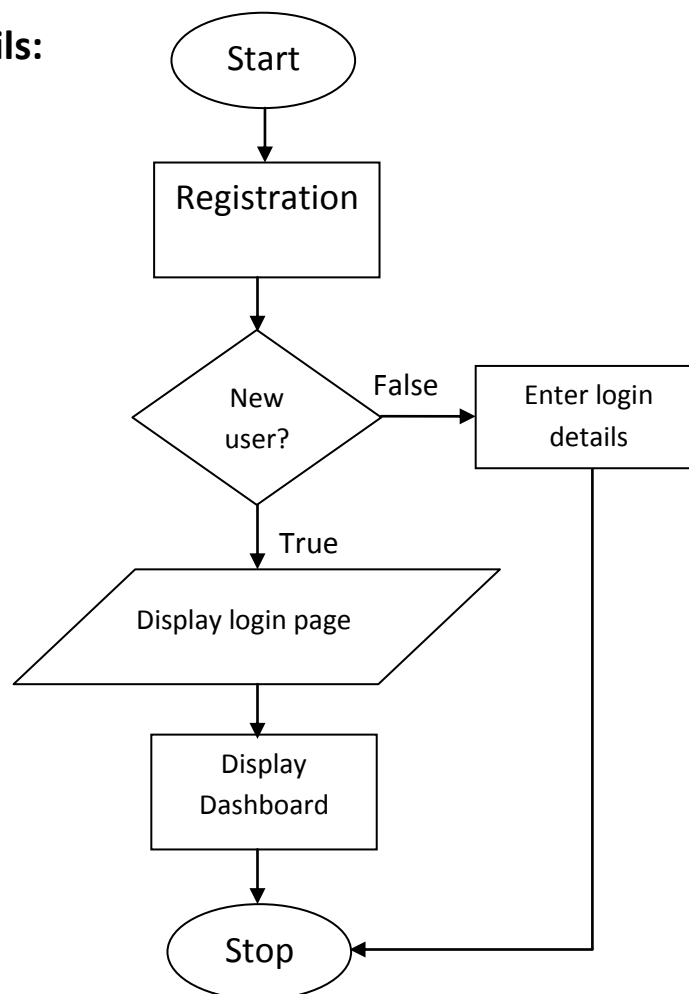**Output:** Admin can manage check-in and check-out process.

5.3.2 Users
Register:

Input: user_email, user_pass;

Procedural Details:

```
                    ( Start )
                       |
                       v
                +-------------+
                | Registration|
                +-------------+
                       |
                       v
                   < New      > -- False --> +-------------+
                   < user?    >              | Enter login |
                       |                      |   details   |
                      True                    +-------------+
                       |                            |
                       v                            |
                / Display login page /              |
                       |                            |
                       v                            |
                +-------------+                     |
                |   Display   |                     |
                |  Dashboard  |                     |
                +-------------+                     |
                       |                            |
                       v                            |
                   ( Stop ) <------------------------
```
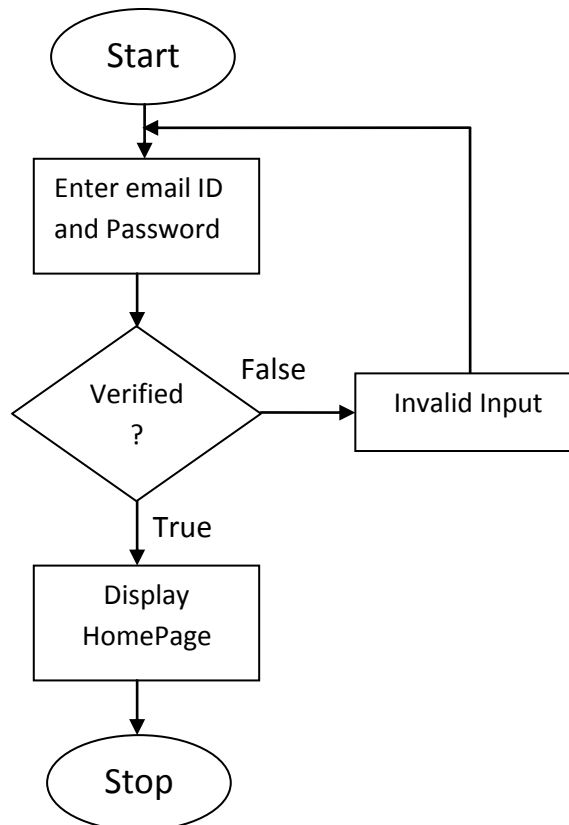
File I/O Interface: Registration page.

Output: User gives their required details to register.

Login:

33

Govinda Dasa College, Surathkal

PARK BOOK

Input: user_email, user_pass;

Procedural Details:

```
                    ( Start )
                        |
                        v
              +--------------------+
              | Enter email ID     | <----+
              | and Password       |      |
              +--------------------+      |
                        |                 |
                        v                 |
                     / Verified \  False  +-------------------+
                    <     ?      >------->|   Invalid Input   |
                     \         /          +-------------------+
                        |
                        | True
                        v
              +--------------------+
              |     Display        |
              |    HomePage        |
              +--------------------+
                        |
                        v
                    ( Stop )
```

File I/O Interface: User page.

Output: Email id and password will be checked for validity. If it is valid, user will be directed to the homepage.

Verify:

Input: phone_no, otp;

File I/O Interface: User page.

Output: User's phone number will be verified for further booking or communication.

Procedural Details:

Govinda Dasa College, Surathkal

PARK BOOK
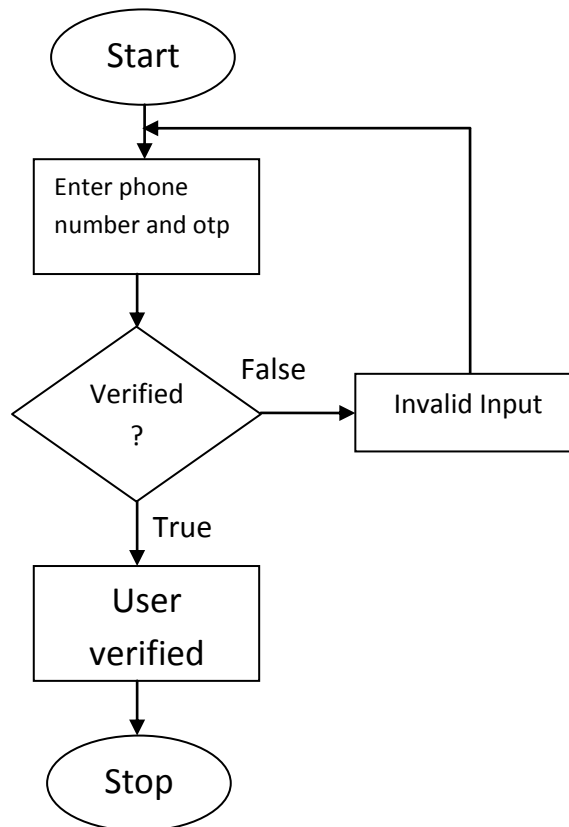
Input details for booking:

Input: vehicle_type, start_date, start_time, end_date, end_time, vehicle_no;

File I/O Interface: Booking page.

Output: User can enter necessary details for booking and it has to be checked for validity. If it is valid, slots will be displayed.

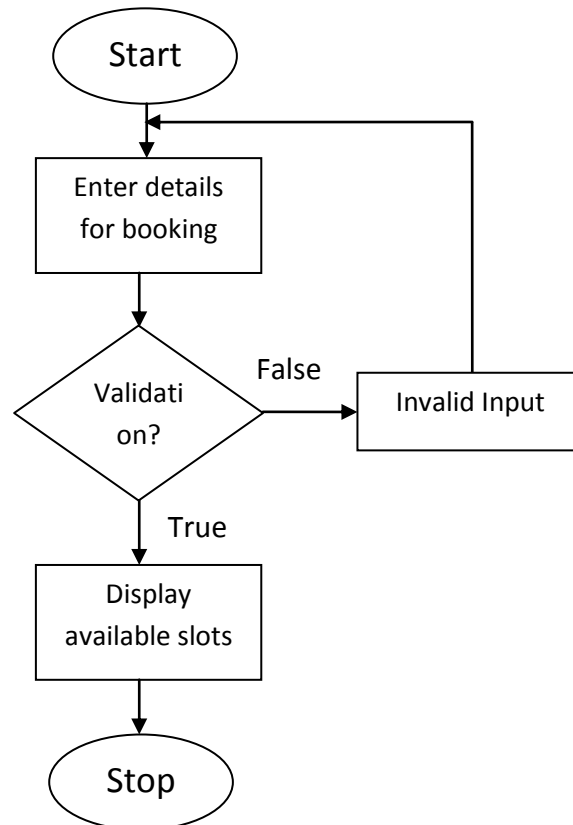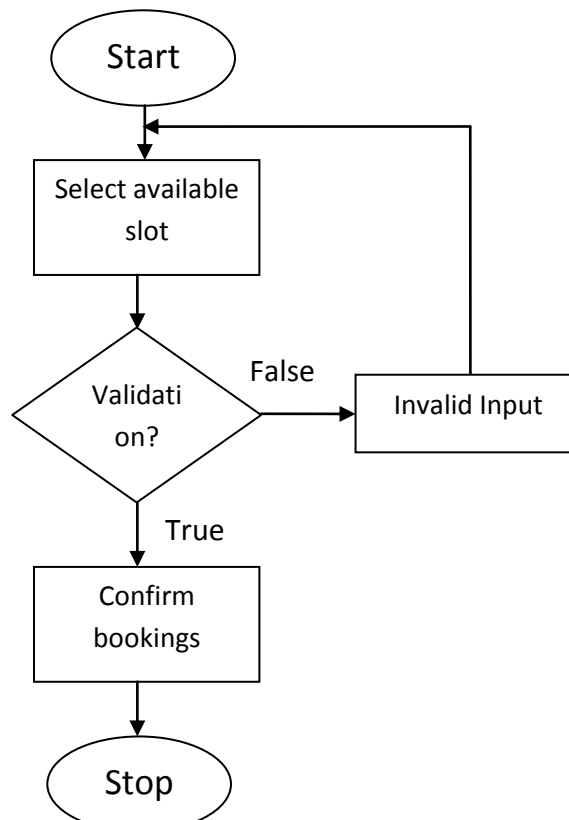**Procedural Details:**

PARK BOOK
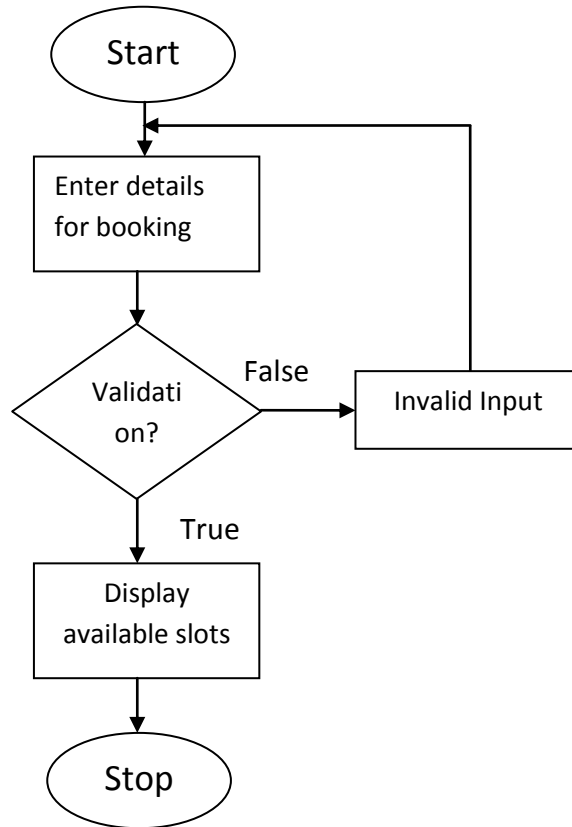
Fetch available slots:

Input: vehicle_type, start_date, start_time, end_date, end_time, vehicle_no;

File I/O Interface: Slots page.

Output: User can enter necessary details for booking and it has to be checked for validity. If it is valid, slots will be displayed.

Procedural Details:

PARK BOOK

Select slots and confirm booking:

Input: select_slot;

Procedural Details:

Govinda Dasa College, Surathkal

PARK BOOK

File I/O Interface: Confirm page.

Output: User can select available slots and make booking.

Check-in process:

Input: otp;

Procedural Details:

```
        ( Start )
            │
            ▼
    ┌──────────────────┐
    │ User provides    │◄─────────────┐
    │ check-in otp to  │              │
    │ admin            │              │
    └──────────────────┘              │
            │                         │
            ▼                         │
         ◇ System      ─── False ──►  ┌─────────────┐
          validates                   │ Invalid OTP │
          OTP? ◇                      └─────────────┘
            │
          True
            │
            ▼
   ┌──────────┐  False   ◇ Checks slot
   │ Assign   │◄──────── is checked-
   │ next     │          out or not? ◇
   │ available│               │
   │ slot     │             True
   └──────────┘               │
        │                     ▼
        │              ┌──────────────┐
        └─────────────►│ Check-in     │
                       │ successful   │
                       └──────────────┘
                              │
                              ▼
                          ( Stop )
```

PARK BOOK
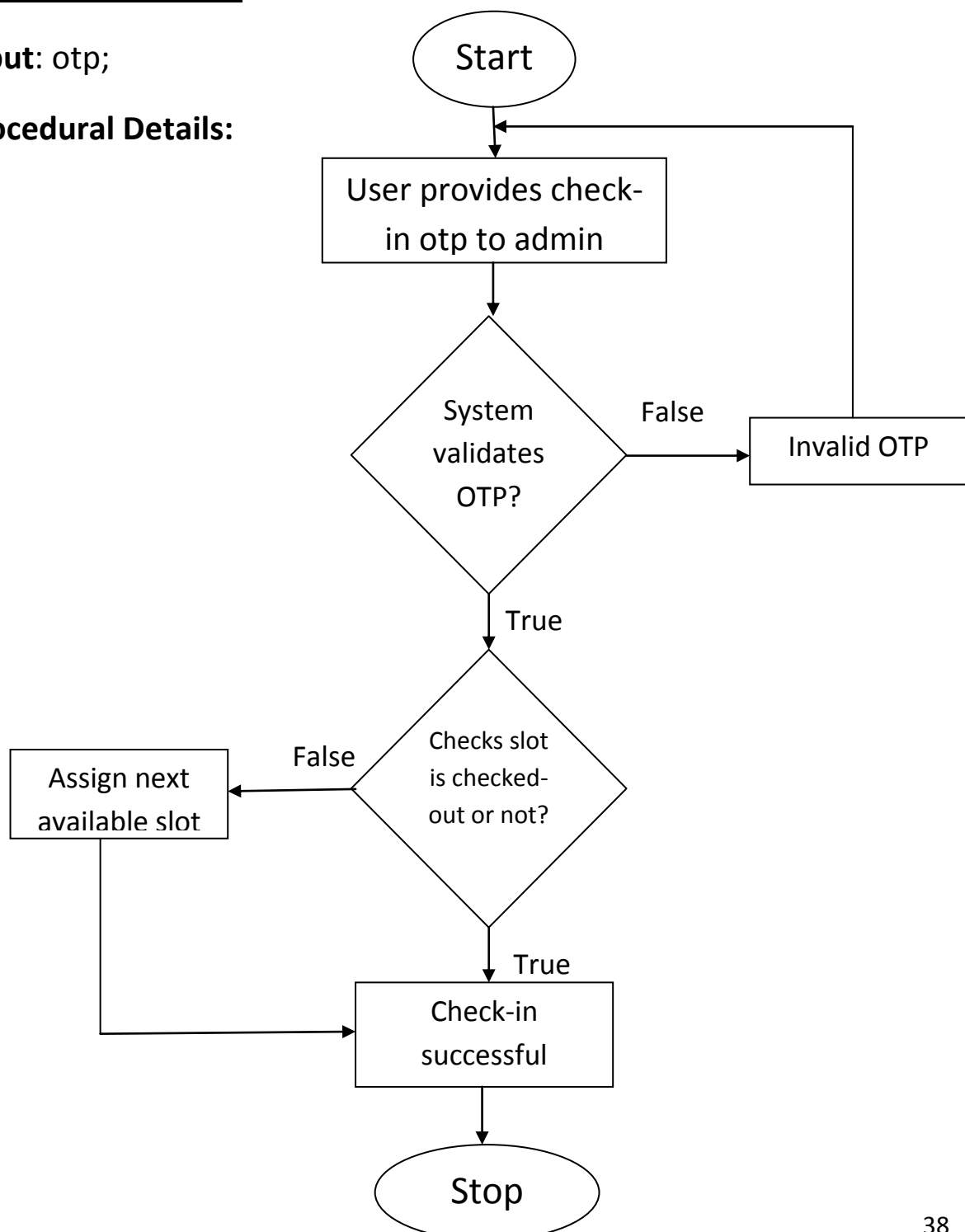
File I/O Interface: Check-in page.

Output: System validates OTP. If it is valid, check-in successful and also system validates whether specified slot is checked-out or not. If it is not checked-out, system assigns next available slots.

Check-out process:

Input: otp;

Procedural Details:

```
                        ┌─────────┐
                        │  Start  │
                        └─────────┘
                             │
                ┌────────────────────────┐
                │  User provides check-   │◄──────┐
                │   in otp to admin       │       │
                └────────────────────────┘       │
                             │                    │
                          ╱     ╲                 │
                        ╱  System  ╲    False   ┌────────────┐
                       ╱  validates  ╲─────────►│ Invalid OTP│
                        ╲   OTP?    ╱            └────────────┘
                          ╲      ╱
                             │ True
                ┌────────────────────────┐
                │     Check-out           │
                │     successful          │
                └────────────────────────┘
                             │
                        ┌─────────┐
                        │  Stop   │
                        └─────────┘
```
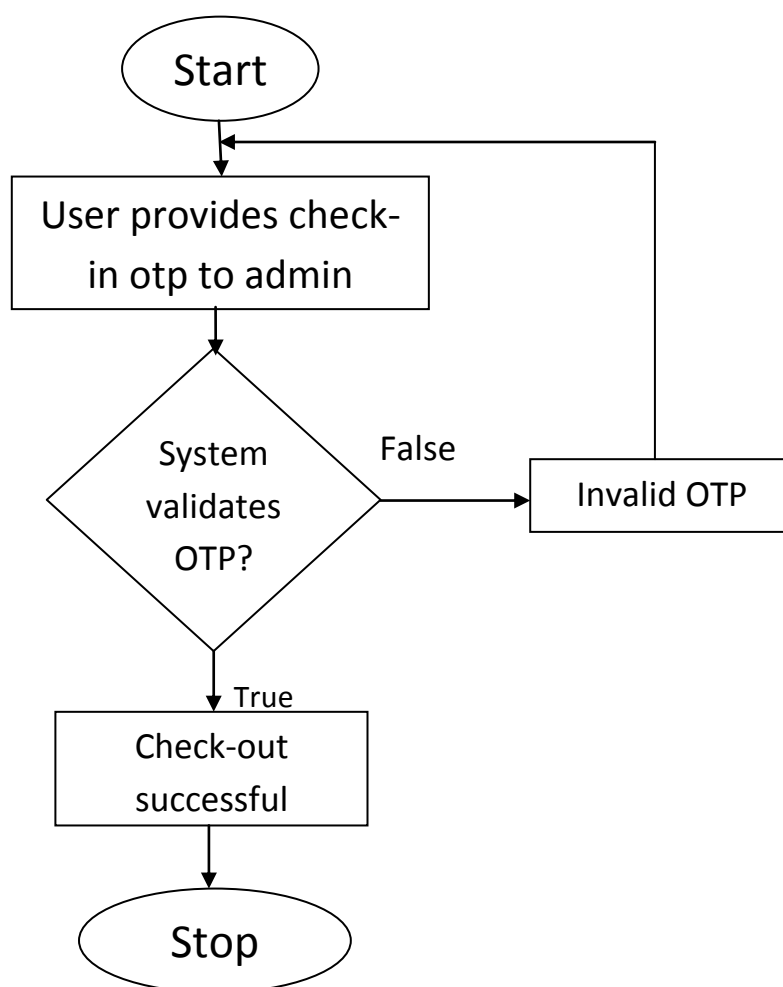
File I/O Interface: Check-out page.

Output: System validates OTP. If it is valid, check-out successful.

TESTING

6. TESTING

6.1 Introduction (about Testing)

Software testing is the process used to help identify the correctness, completeness, security and quality of developed computer software. This includes the process of executing the program or application with the intent of finding errors. Quality is not an absolute; it is value to some person. With that in mind testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behaviour of the product against a specification. The testing phase consists of evaluating the software that has been developed in order to conform that it produces the output required in a safe and efficient manner.

In this phase inherent errors that occur, must be handled and the user should be informed so that he/she can follow the guidelines and instructions and get around the error and obtain the output. During testing, the program to be tested is executed with a set of test cases and the output of the program for the test cases is evaluated to determine if the program is performing as expected. Due to its approach, dynamic testing can only ascertain the presence of errors in the program the exact nature of the errors is not usually decided by testing. Testing forms the first step in determining the errors in a program.

Clearly the success of testing in revealing errors in programs depends critically on the test cases. The program to be tested is executed with a set of test cases and the output of the program for the test cases are evaluated to determine if the programming is performing as expected. Testing forms the first step in determining errors in a program. The success of testing in revealing errors in programs depends critically on the test cases.

6.2 Test Reports:

- Unit Testing
- Integration Testing

- System testing

6.2.1 Unit Testing:

In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage producers, are tested to determine if they are fit to use. Intuitively, one can view a unit as the smallest testable part of an application, In procedural programming a unit could be an entire module but is more commonly an individual function or procedure. In object oriented programming a unit is often an entire interface, such as class, but could be an individual method. Unit tests are created by programmers or occasionally by white box testers during the development process.

6.2.2 Integration Testing:

The purpose of integration testing is to verify functional performance. And reliability requirements placed on major design items. These design items. i.e. assemblages (or group of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test that all components with in assemblages interact correctively, for example across producers call of procedures activation, and this is done after testing individual modules, i.e. unit testing.

6.2.3 System Testing:

A system testing of software or hardware is testing conducted on a complete, integrated system to evaluate system's compliance with its specified requirements. System testing falls within the scope of black box testing, and such as, should require no knowledge of the inner design of the integrated software components that have successfully passed integration testing and also the software components itself integrated with any applicable hardware system(s). The purpose of Integration resting is to detect any inconsistencies between software units that are integrated together called assemblages) or between any of the assemblages and the hardware, System is more limited

type of testing, it seeks to detect defects both within the inter-assemblages and also within the system as whole.

6.3 Testing for user interface:

6.3.1 Admin
Login form:

Sl. No.	Test Condition	Expected Result	Result
1	If admin clicks on login button without entering email or password	Please fill out this field	Successful
2	If Email field is blank but password is entered	Please fill out this field	Successful
3	If Password is blank but Email is entered	Please fill out this field	Successful
4	If Email or password is incorrect	Invalid email and password	Successful
5	If valid email and password is entered	Displays home page	Successful

Home Page:

Sl. No.	Test Condition	Expected Result	Result
1	If admin clicks on "Update Prices" in the menu	Update Price component will be displayed contains slots prices	Successful
2	If admin clicks on "Admin History" in the menu	Admin history page will be displayed contains all the bookings	Successful
3	If admin clicks on "Slots" in the menu	Slot page will be displayed	Successful

Update Price:

Sl. No.	Test Condition	Expected Result	Result

PARK BOOK

1	If admin clicks on "Fetch latest price" button	Latest price will be displayed	Successful
2	If admin enters on negative values for car and bike prices	Please enter positive number	Successful
3	If admin updates car or bike price	Price updated successfully	Successful

Add/Remove slots:

Sl. No.	Test Condition	Expected Result	Result
1	If admin enters already existing slot id and presses add slot button	Slot is already present	Successful
2	If admin enters not available slot id and presses remove slot button	There is no such slot exist	Successful
3	If admin enters valid slot id and presses add slot button	New slot added successfully	Successful
4	If admin enters existing slot id and presses remove slot button	Slot removed successfully	Successful

Admin History:

Sl. No.	Test Condition	Expected Result	Result
1	If admin presses "Refresh" button	History will be displayed	Successful
2	If admin searches by vehicle number or user name	Corresponding information will be displayed	Successful
3	If admin enters invalid check-in OTP	Wrong OTP	Successful
4	If admin enters the correct check-in OTP	Checked-in Successfully	Successful
5	If admin enters invalid check-out OTP	Wrong OTP	Successful
6	If admin enters the correct check-out OTP	Checked-out Successfully	Successful

Govinda Dasa College, Surathkal

PARK BOOK

6.3.2 User Registration form:

Sl. No.	Test Condition	Expected Result	Result
1	If user clicks on register button without entering email or password	Please fill out this field	Successful
2	If Email field is blank but password is entered	Please fill out this field	Successful
3	If Password is blank but Email is entered	Please fill out this field	Successful
4	If password is incorrect format	Invalid password format	Successful
5	User can register directly using google account	Displays home page	Successful
6	If valid email and password is entered	Displays home page	Successful

Login form:

Sl. No.	Test Condition	Expected Result	Result
1	If user clicks on login button without entering email or password	Please fill out this field	Successful
2	If Email field is blank but password is entered	Please fill out this field	Successful
3	If Password is blank but Email is entered	Please fill out this field	Successful
4	If Email or password is incorrect	Invalid email and password	Successful
5	If valid email and password is entered	Displays home page	Successful
6	User can login using google account	Displays home page	Successful

PARK BOOK

User Home Page:

Sl. No.	Test Condition	Expected Result	Result
1	If user clicks on "Book" from the menu	Displays a profile page	Successful
2	If user clicks on "Admin" from the menu	Displays access denied	Successful
3	If user clicks on "Admin history" from the menu	Displays access denied	Successful
4	If user clicks on "Admin history" from the menu	Displays verify page	Successful
5	If user clicks on "History" from the menu	Displays user's booking	Successful
6	If user clicks on "Log Out" from the menu	User logged out from the system	Successful

Booking Page:

Sl. No.	Test Condition	Expected Result	Result
1	If user enters past date or time	Booking times must be in the future	Successful
2	If user misses any of the input field	Please fill all the fields	Successful
3	If user enters invalid vehicle number	Please enter valid vehicle number	Successful
4	If user clicks on already booked slot	Slot is already occupied	Successful
5	If user clicks on unoccupied slot	Slot will be selected	Successful
6	If user clicks on "Book Now" button	Booking information will be displayed	Successful
7	If user clicks on "Cancel" button	Go back to the booking page	Successful
8	If user clicks on "Pay and Confirm" button	Booking placed successfully	Successful

Confirm Page:

Govinda Dasa College, Surathkal

PARK BOOK

Sl. No.	Test Condition	Expected Result	Result
1	If user clicks on "Home" button	Redirected to home page	Successful
2	If user clicks on "View History" button	History page will be displayed	Successful

Verify Page:

Sl. No.	Test Condition	Expected Result	Result
1	If user enters invalid OTP	Invalid OTP	Successful

6.4 System Testing:

System Testing is the testing of a complete and fully integrated software product. Usually, software is only one element of a larger computer-based system. Ultimately, software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

6.4.1 System Testing Tables

Sl. No.	Test Condition	Test Report
1	System run procedure	Successful
2	File I/O operation	Successful
3	Database communication	Successful
4	Server/client interaction	Successful
5	Memory usage	Normal
6	System processor usage	Normal
7	Authentication/Authorization	Successful

CODING

7. CODING

7.1 Introduction (about Coding)

The goal of coding or programming phase is to translate the system design, produced during the designing phase, into code in a given programming language which can be executed by a computer and that performs the computation specified by the design. During the implementation, it should be kept in mind that the programs should not be constructed so that they are easy to write, but that they are easy to read and understand.

7.2 Program Code Listing:

Backend and Database connection (server.js):

```
const express = require('express');

const cors = require('cors');

const { MongoClient, ObjectId } = require('mongodb');

require('dotenv').config()

const app = express();

app.use(cors());

app.use(express.json());


// MongoDB connection string

const uri =   process.env.MONGO;

const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });


// Database and collections

let db, bookingsCollection, slotsCollection;


// Connect to DB

async function connectToDB() {

  await client.connect();
```

Govinda Dasa College, Surathkal

PARK BOOK

```javascript
  db = client.db('parkingApp');

  bookingsCollection = db.collection('bookings');

  slotsCollection = db.collection('slots');

}


// Check if a booking time overlaps with the requested time range

function isTimeOverlap(bookingStart, bookingEnd, queryStart, queryEnd) {

    return bookingStart < queryEnd && bookingEnd > queryStart;

}


// Update slot availability based on a given time range

app.get('/api/parkingSlots', async (req, res) => {

  const { bookedFrom, bookedTill, type } = req.query; // Include type in your query

  if (!bookedFrom || !bookedTill || !type) {

      return res.status(400).json({ error: "Please provide 'bookedFrom', 'bookedTill', and 'type'
parameters." });

  }

  const availableSlots = await getAvailableSlots(bookedFrom, bookedTill, type);

  res.json(availableSlots);

});


async function getAvailableSlots(bookedFrom, bookedTill, type) {

    const slots = await slotsCollection.find({ type }).toArray(); // Filter by type

    const bookings = await bookingsCollection.find().toArray();

    const requestedStartTime = new Date(bookedFrom);

    const requestedEndTime = new Date(bookedTill);


    slots.forEach(slot => slot.isOccupied = false);


    bookings.forEach(booking => {

      const bookingStartTime = new Date(booking.bookedFrom);

      const bookingEndTime = new Date(booking.bookedTill);
```

PARK BOOK

```
        if (isTimeOverlap(bookingStartTime, bookingEndTime, requestedStartTime,
requestedEndTime)) {

            const slot = slots.find(s => s.slotId === booking.slotId);

            if (slot) {

                slot.isOccupied = true;

            }

        }

    });


    return slots;

}

function generateOTP(length = 4) {

    let otp = '';

    for(let i = 0; i < length; i++) {

        otp += Math.floor(Math.random() * 10).toString();

    }

    return otp;

}




// Endpoint to create a new booking if the slot is available

app.post('/api/bookings', async (req, res) => {

 const { userId, vehicleType, amount, slotId, bookedFrom, bookedTill, paymentDetails
,vehicleNumber} = req.body;

 const slots = await getAvailableSlots(bookedFrom, bookedTill,vehicleType);

 const slot = slots.find(s => s.slotId === slotId && !s.isOccupied);




 if (!slot) {

   return res.status(400).json({ error: 'Requested slot is not available for the selected time.'
});

 }
```

Govinda Dasa College, Surathkal

PARK BOOK

```javascript
  // Mark the slot as occupied for the new booking
  slot.isOccupied = true;
  const newBooking = {
    userId, vehicleType, amount, slotId, bookedFrom, bookedTill,
    bookingId: `booking${new ObjectId()}`,
    paymentDetails,
    vehicleNumber,
    isCheckedIn: false,
    isCheckedOut: false
  };


  await bookingsCollection.insertOne(newBooking);
  await slotsCollection.updateOne({ slotId }, { $set: { isOccupied: true } });


  res.status(201).json(newBooking);
});


// Endpoint to fetch booking history for a specific user
app.get('/api/userBookings', async (req, res) => {
    const { email } = req.query;
    if (!email) {
        return res.status(400).json({ error: "Email parameter is required." });
    }


    const userBookings = await bookingsCollection.find({ userId: email }).toArray();
    res.json(userBookings);
});


// Endpoint to fetch all bookings of all users
app.get('/api/allBookings', async (req, res) => {
  const bookings = await bookingsCollection.find().toArray();
  res.json(bookings);
```

Govinda Dasa College, Surathkal

PARK BOOK

```javascript
});
app.post('/api/check-in', async (req, res) => {
  const { bookingId, vehicleNumber } = req.body;


  try {
    const booking = await bookingsCollection.findOne({ _id: ObjectId(bookingId),
vehicleNumber });
    if (!booking) {
      return res.status(404).json({ error: 'Booking not found or vehicle number mismatch.'
});
    }


//      // Verify if the previous booking has checked out
// const lastBooking = await bookingsCollection.find({ slotId: booking.slotId })
//    .sort({ bookedTill: -1 }) // Sort by 'bookedTill' in descending order
//    .limit(1) // Limit to get only the first document after sorting
//    .next(); // Get the first document from cursor


// if (lastBooking && !lastBooking.isCheckedOut) {
//    // Latest booking has not checked out yet
//    return res.status(403).json({ error: 'Previous booking has not checked out yet.' });
// }


    const otp = generateOTP();  // Assume generateOTP is already defined


    // Mark this booking as checked in
    await bookingsCollection.updateOne(
      { _id: ObjectId(bookingId) },
      { $set: { isCheckedIn: true } }
    );


    res.json({ message: 'Check-in successful', slotId: booking.slotId, otp });
```

Govinda Dasa College, Surathkal

PARK BOOK

```javascript
    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Internal server error.' ,error});

    }

});


app.post('/api/check-out', async (req, res) => {

  const { bookingId, otp } = req.body;


  try {

    const booking = await bookingsCollection.findOne({ _id: ObjectId(bookingId), otp });

    if (!booking || !booking.isCheckedIn) {

      return res.status(404).json({ error: 'Invalid OTP or booking ID, or check-in not
completed.' });

    }


    // Update the booking and slot status on checkout

    await bookingsCollection.updateOne(

      { _id: ObjectId(bookingId) },

      { $set: { isCheckedOut: true } }

    );

    await slotsCollection.updateOne(

      { _id: booking.slotId },

      { $set: { isOccupied: false, currentVehicle: null } }

    );

    res.json({ message: 'Check-out successful, slot now available.' });


  } catch (error) {

    console.error(error);

    res.status(500).json({ error: 'Internal server error during check-out.' });

  }
```

PARK BOOK

```
});


// Add a Slot
app.post('/api/slots', async (req, res) => {
    const { slotId, type } = req.body;
    try {
        const newSlot = { slotId, type, isOccupied: false };
        await slotsCollection.insertOne(newSlot);
        res.status(201).json({ message: 'New slot added successfully.', newSlot });
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Failed to add new slot.' });
    }
});


// Remove a Slot
app.delete('/api/slots/:slotId', async (req, res) => {
    const { slotId } = req.params;
    try {
        const result = await slotsCollection.deleteOne({ slotId });
        if (result.deletedCount === 0) {
            return res.status(404).json({ error: 'Slot not found.' });
        }
        res.json({ message: 'Slot removed successfully.' });
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Failed to remove slot.' });
    }
});
const PORT = process.env.PORT || 3001;
connectToDB().then(() => {
 app.listen(PORT, () => {
```

Govinda Dasa College, Surathkal

PARK BOOK

```
  console.log(`Server running on http://localhost:${PORT}`);
 });
}).catch(console.error);
```

Admin (admin.js):

```
import React from 'react';
import UpdatePrices from './updateprice';
import UpdateSlots from './updateslots';
import Adminhist from './adminhist';
import Accessdenied from './accessdenied';
import Current from './current';
import { useState, useEffect } from 'react';


import { useAuth0 } from '@auth0/auth0-react';
const Admin = () => {


  const { user } = useAuth0();
  const [page, setPage] = useState("1");
  const [admin, setadmin] = useState("false");
  const [loading, setLoading] = useState();




  useEffect(() => {
    if (user) {
      setadmin(user.email === "deveshpoojary@gmail.com" || user.email ===
"tharunrai14@gmail.com");
      setLoading(false);
    }
    else {
```

```
      setLoading(true);

   }

   // Check if the user is an admin only when the component mounts or user changes


 }, [user]); // Depend on user.email


 function handle(e) {

   setPage(e.target.name);


 }



 return (

   < > <div className='bg-primary min-h-screen'>

     {admin ? <> <div className='bg-secondary flex px-4 py-4 shadow-lg border-b border-gray-500' >

       <button className='bg-white text-black border border-white font-bold py-2 px-4 rounded-md hover:bg-black hover:text-white mr-2' name='1' onClick={handle}>

         Update Prices

       </button>


       <button className='bg-white text-black border border-white font-bold py-2 px-4 rounded-md hover:bg-black hover:text-white mr-2' name="2" onClick={handle}>

         Admin History

       </button>



       <button className='bg-white text-black border border-white font-bold py-2 px-4 rounded-md hover:bg-black hover:text-white mr-2' name="3" onClick={handle}>

         Slots        </button>


       <button className='bg-white text-black border border-white font-bold py-2 px-4 rounded-md hover:bg-black hover:text-white mr-2' name="4" onClick={handle}>

         Current        </button>
```

```jsx
        </div>
            {page === "1" ? <UpdatePrices /> : page === "2" ? <Adminhist /> : page==="3"?
<UpdateSlots />:<Current/>  }
        </>


            : loading ? <div className='text-white font-bold'>Loading...</div> :


        <div>
          {/* <h1 className='text-white font-bold'>Access Denied</h1> */}
          <Accessdenied/>
        </div>}


    </div>
    </>
  );
};


export default Admin;
```

Update slot Page (updateslot.js):

```jsx
import React, { useState } from 'react';


const UpdateSlots = () => {
   const [slotId, setSlotId] = useState(0);
   const [vehicleType, setVehicleType] = useState('car');
   const [numSlots, setNumSlots] = useState(1);
   const [operation, setOperation] = useState('add'); // 'add' or 'remove'
   const [error,setError]=useState("")
   const [message, setMessage] = useState('');
```

PARK BOOK

```javascript
const handleAddSlots = async () => {
  if(!(slotId<0) && !(numSlots<=0))
  {

    const currentSlotId =slotId ;
    const slotData = {
      slotId: parseInt(currentSlotId),
      type: vehicleType,
      numberofslots:parseInt(numSlots)
    };
    try {
      const response = await fetch('https://park-book-
9f9254d7f86a.herokuapp.com/api/slots', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(slotData)
      });
      const data = await response.json();
      console.log('Slot added:', data);
      setMessage("Slot added successfully")
    } catch (error) {
      console.error('Failed to add slot:', error);

  }}
  else{
    setError("  Invalid data entered");
  }
};
```

```
const handleRemoveSlot = async () => {

  if(!(slotId<=0) && !(numSlots<=0)){

      const currentSlotId =slotId ;

    const slotData = {

      slotId: parseInt(currentSlotId),


    };

  try {

    const response = await fetch(`https://park-book-
9f9254d7f86a.herokuapp.com/api/slotsdel`, {

      method: 'POST',

      headers: {

          'Content-Type': 'application/json'

        },

        body: JSON.stringify(slotData)

    });

    const data = await response.json();

    console.log('Slot removed:', data);

    setMessage("Slot removed successfully")

  } catch (error) {

    console.error('Failed to remove slot:', error);

  }

  }

  else{

    setError("  Invalid data entered");

  }


};


const handleSubmit = (e) => {

  e.preventDefault();

  if (operation === 'add') {
```

Govinda Dasa College, Surathkal

```
        handleAddSlots();

    } else {

        handleRemoveSlot();

    }

    };


    return (

        <div className="container bg-secondary mx-auto px-4 shadow-lg shadow-gray-500
rounded-lg py-4 mt-6">

            <h1 className="text-4xl py-2 mt-4 font-bold text-white l-border">Update Parking
Slots</h1>

            <form onSubmit={handleSubmit} className="mt-4">

                <div className="mb-4">

                    <label htmlFor="slotId" className="block text-sm font-medium text-
white">Starting Slot ID</label>

                    <input

                        type="number"

                        id="slotId"

                        value={slotId}

                        onChange={e => setSlotId(e.target.value)}

                        className="bg-primary picker form-input hover:border-cyan-500 mt-1 block
w-full px-3 py-2 border border-gray-500 rounded-md bg-transparent text-white"


                    />

                </div>

                <div className="mb-4">

                    <label htmlFor="vehicleType" className="block text-sm font-medium text-
white">Vehicle Type</label>

                    <div className="flex">

                        <div className="mr-4">

                            <input

                                type="radio"

                                id="car"

                                value="car"
```

Govinda Dasa College, Surathkal

```
                checked={vehicleType === 'car'}

                onChange={e => setVehicleType(e.target.value)}

                className="mr-2"

              />

              <label htmlFor="car" className="text-white">Car</label>

            </div>

            <div>

              <input

                type="radio"

                id="bike"

                value="bike"

                checked={vehicleType === 'bike'}

                onChange={e => setVehicleType(e.target.value)}

                className="mr-2"

              />

              <label htmlFor="bike" className="text-white">Bike</label>

            </div>

          </div>

        </div>

        {operation==='add'?<div className="mb-4">

  <label htmlFor="numSlots" className="block text-sm font-medium text-white">Number
of Slots</label>

  <div className="flex items-center mt-1">

    <button

      type="button"

      onClick={() => setNumSlots(Math.max(1, numSlots - 1))}

      className="bg-primary border border-gray-500 hover:border-cyan-500 rounded-l-
md px-3 py-2 text-white"

    >

      -

    </button>

    <input

      type="number"
```

Govinda Dasa College, Surathkal

```
      id="numSlots"

      min={1}

      value={numSlots}

      onChange={e => setNumSlots(Math.max(1, Number(e.target.value)))}

      className="bg-primary picker block w-16 px-3 py-2 border-t border-b border-gray-
500 text-center text-white bg-transparent"

      required

    />

    <button

      type="button"

      onClick={() => setNumSlots(numSlots + 1)}

      className="bg-primary border border-gray-500 hover:border-cyan-500 rounded-r-
md px-3 py-2 text-white"

    >

      +

    </button>

  </div>

</div>

:""}

        <div className="mb-4">

  <label className="block text-sm font-medium text-white">Operation</label>

  <div className="mt-1">

    <label className="inline-flex items-center text-white">

      <input

        type="radio"

        value="add"

        checked={operation === 'add'}

        onChange={e => setOperation(e.target.value)}

        className="bg-primary picker border-gray-500 hover:border-cyan-500 rounded-
md bg-transparent text-white"

      />

      <span className="ml-2">Add Slots</span>

    </label>
```

Govinda Dasa College, Surathkal

```
    <label className="inline-flex items-center text-white ml-4">

      <input

        type="radio"

        value="remove"

        checked={operation === 'remove'}

        onChange={e => setOperation(e.target.value)}

        className="bg-primary picker border-gray-500 hover:border-cyan-500 rounded-
md bg-transparent text-white"

      />

      <span className="ml-2">Remove Slot</span>

    </label>

  </div>

</div>


      <button

        type="submit"

        className="bg-white hover:bg-black hover:text-white text-black border border-
white font-bold py-2 px-4 rounded"

      >

        {operation === 'add' ? 'Add Slots' : 'Remove Slot'}

      </button>

    </form>

    {error}{message && <p className="mt-4 text-green-600">{message}</p>}

  </div>

  );

};


export default UpdateSlots;
```

Admin History Page (adminhist.js):

```
import React, { useEffect, useState } from 'react';

import { useAuth0 } from '@auth0/auth0-react';
```

Govinda Dasa College, Surathkal

PARK BOOK

```
import { Alert } from '@mui/material';

import LoadingAnimation from '../steering';

import Accessdenied from './accessdenied';


const Adminhist = () => {

    const { user } = useAuth0();

    const [bookings, setBookings] = useState([]);

    const [count, setCount] = useState(0);

    const [otp, setOtp] = useState('');

    const [otp2, setOtp2] = useState('');

    const [loading, setLoading] = useState(false);

    const [searchTerm, setSearchTerm] = useState('');

    const [searchResults, setSearchResults] = useState([]);

    const [error, setError] = useState(null);

    const [mess, setMess] = useState('');

    const [adminhist, setAdminhist] = useState('false');



    useEffect(() => {

        fetchUserBookings();

    }, [count]);


    const fetchUserBookings = async () => {

        try {

            setLoading(true);

            const response = await fetch(`https://park-book-
9f9254d7f86a.herokuapp.com/api/allBookings`);

            if (response.ok) {

                const fetchedBookings = await response.json();

                fetchedBookings.sort((a, b) => new Date(b.bookedFrom) - new
Date(a.bookedFrom));

                setBookings(fetchedBookings);

                setSearchResults(fetchedBookings);
```

```
            setLoading(false);
        } else {
            throw new Error('Failed to fetch bookings');
        }
    } catch (error) {
        console.error('Error fetching user bookings:', error);
        setLoading(false);
    }
};


const checkin = async (booking) => {
    try {
        const checkInTime = new Date();
        checkInTime.setHours(checkInTime.getHours() + 5);
        checkInTime.setMinutes(checkInTime.getMinutes() + 30);
        const trimdaytime = checkInTime.toISOString().slice(0, 19).replace('T', ' ');
        const url = `https://park-book-9f9254d7f86a.herokuapp.com/api/check-in`;
        const response = await fetch(url, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                bookingId: booking.bookingId,
                checkInTime: trimdaytime,
                checkinotp: otp
            })
        });

        const result = await response.json();
        switch (response.status) {
            case 200:
```

```
      console.log("success");

      setMess("Checked in successfully");

      setCount(count + 1);

      break;

   case 209:

      console.log("New Slot Assigned", result.newSlotId);

      setMess(`New Slot Assigned: ${result.newSlotId}`);

      setCount(count + 1);

      break;

   case 405:

      console.log("Wrong OTP");

      setError("Wrong OTP");

      setCount(count + 1);

      break;

   case 404:

      console.log("Booking not found");

      setError("Booking not found");

      setCount(count + 1);

      break;

   case 401:

      console.log("Unauthorized");

      setError("Unauthorized");

      setCount(count + 1);

      break;

   case 408:

      console.log("Cannot check in now, check in after booking time");

      setError("Cannot check in now, check in after booking time");

      setCount(count + 1);

      break;

   case 406:

      console.log("Already checked in");

      setError("Already checked in");
```

67

```
            setCount(count + 1);

            break;

          case 500:

            console.log("Server error");

            setError("Server error");

            break;

          case 201:

            console.log("Check in successful");

            setMess("Checked in successfully");

            setCount(count + 1);

            break;

          default:

            console.log("error");

        }

      } catch (error) {

        console.log('Error checking in:', error);

        setError(error.message);

      }

    };


    const checkout = async (booking) => {

      try {

        const checkOutTime = new Date();

        checkOutTime.setHours(checkOutTime.getHours() + 5);

        checkOutTime.setMinutes(checkOutTime.getMinutes() + 30);

        const trimdaytime = checkOutTime.toISOString().slice(0, 19).replace('T', ' ');

        const url = `https://park-book-9f9254d7f86a.herokuapp.com/api/check-out`;

        const response = await fetch(url, {

          method: 'POST',

          headers: {

            'Content-Type': 'application/json',

          },
```

Govinda Dasa College, Surathkal

```
            body: JSON.stringify({

                bookingId: booking.bookingId,

                checkOutTime: trimdaytime,

                checkoutotp: otp2

            })

        });

        const result = await response.json();

        if (response.status === 200) {

            console.log("success", result);

            setMess("Checked out successfully");

            setCount(count + 1);

        }

        else {

            console.log("error", result);

            setError(result.error);

        }



    } catch (error) {

        console.error('Error checking out:', error);

        setError(error.message);

    }

};


const handleOtpChange = (otp) => {

    setOtp(otp);

};


const handleOtpChange2 = (otp) => {

    setOtp2(otp);

};
```

PARK BOOK

```javascript
const isBookingExpired = (bookedTill) => {

  const now = new Date();

  const tillDate = new Date(bookedTill);

  return now > tillDate;

};


useEffect(() => {

  if (searchTerm === "") {

    setBookings(searchResults);

  } else {

    const results = bookings.filter(booking =>

      (booking.vehicleNumber && booking.vehicleNumber.includes(searchTerm)) ||

      (booking.userId &&
booking.userId.toLowerCase().includes(searchTerm.toLowerCase()))

    );

    setBookings(results);

  }

}, [searchTerm]);


useEffect(() => {

  if (user) {

    setAdminhist(user.email === "deveshpoojary@gmail.com" || user.email ===
"tharunrai14@gmail.com");

    setLoading(true);

  }

  else {

    setLoading(false);

  }

  // Check if the user is an admin only when the component mounts or user changes


}, [user]); // Depend on user.email


return (<>{adminhist ?
```

PARK BOOK

```
<div className='bg-primary text-white min-h-screen'>

    <span className="flex text-2xl font-bold mb-4 px-2 py-4 border-b border-gray-500
bg-secondary shadow-lg">

        <h1 className='l-border fam'>All Bookings</h1>

        <button onClick={() => setCount(count + 1)} className="bg-white hover:bg-black
hover:text-white text-black font-bold py-1 px-2 border border-white ml-2 rounded-md">

            Refresh

        </button>

        {error && <Alert severity="error" onClose={() => { setError(null) }}>{error}</Alert>}

        {mess && <Alert severity="success" onClose={() => { setMess(null)
}}>{mess}</Alert>}

    </span>


    {!loading ? (

        <div className='px-4 py-4'>

            <div className='overflow-x-auto'>

                <table className="min-w-full border-collapse mt-2 rounded-lg">

                    <thead>

                        <tr>

                            <th colSpan={10} className="border border-gray-300 px-4 py-2">

                                <input

                                    type="text"

                                    placeholder="Search"

                                    onChange={(e) => setSearchTerm(e.target.value)}

                                    className="border border-gray-300 px-4 py-2 rounded-lg bg-
secondary"

                                />

                            </th>

                        </tr>

                        <tr>

                            <th className="border border-gray-300 px-4 py-2">User</th>

                            <th className="border border-gray-300 px-4 py-2">Vehicle no.</th>

                            <th className="border border-gray-300 px-4 py-2">Amount</th>
```

Govinda Dasa College, Surathkal

```
                <th className="border border-gray-300 px-4 py-2">Slot ID</th>

                <th className="border border-gray-300 px-4 py-2">Booked From</th>

                <th className="border border-gray-300 px-4 py-2">Booked Till</th>

                <th className="border border-gray-300 px-4 py-2">Checkin</th>

                <th className="border border-gray-300 px-4 py-2">Checkout</th>

            </tr>

        </thead>

        <tbody>

          {bookings.map(booking => (

            <tr key={booking.bookingId}>

              <td className="border border-gray-300 px-4 py-2">{booking.userId}</td>

              <td className="border border-gray-300 px-4 py-2">{booking.vehicleNumber}</td>

              <td className="border border-gray-300 px-4 py-2">{booking.amount}</td>

              <td className="border border-gray-300 px-4 py-2">{booking.slotId}</td>

              <td className="border border-gray-300 px-4 py-2">{booking.bookedFrom}</td>

              <td className="border border-gray-300 px-4 py-2">{booking.bookedTill}</td>

              <td className="border border-gray-300 px-4 py-2 text-center">

                {!booking.isCheckedIn ? (

                  isBookingExpired(booking.bookedTill) ? (

                    <span className="text-red-600 font-bold">Booking Expired</span>

                  ) : (

                    <>

                      {booking.bookedFrom < new Date().toISOString().slice(0, 19).replace('T', ' ') ? (

                        <p className="text-yellow-400 font-bold">Not Today</p>

                      ) : (

                        <>

                          <input
```

Govinda Dasa College, Surathkal

```
                                    type="number"

                                    placeholder="Enter OTP"

                                    onChange={(e) => handleOtpChange(e.target.value)}

                                    className="placeholder-gray-900 mr-2 text-black
rounded-md py-1"

                                /> 
                                <button

                                    className="bg-cyan-500 hover:bg-black text-white
font-bold py-1 px-2 rounded border border-white mt-2"

                                    onClick={(e) =>{

                                    e.preventDefault();


                                        checkin(booking)}}

                                >

                                    Checkin

                                </button>

                            </>

                        )}

                    </>

                )

            ) : (

                <span className="text-green-600 font-bold">Checked in</span>

            )}

        </td>

        <td className="border border-gray-300 px-4 py-2 text-center">

            {booking.isCheckedIn ? (

                booking.isCheckedOut ? (

                    <span className="text-green-600 font-bold">Checked
out</span>

                ) : (

                    <p className="px-4 py-2">

                        <input
```

```
                            type="number"

                            placeholder="Enter OTP"

                            onChange={(e) => handleOtpChange2(e.target.value)}

                            className="placeholder-gray-900 mr-3 text-black
rounded-md py-1"

                        />
                        <button

                        className="bg-cyan-500 border border-white hover:bg-
black text-white font-bold mt-2 py-1 px-2 rounded"

                            onClick={(e) => {

                                e.preventDefault();

                                checkout(booking)}}

                        >

                            Checkout

                        </button>

                    </p>

                )

            ) : (

                <p className="text-red-600 px-4 py-2 font-bold">Not checked
in</p>

            )}

        </td>

    </tr>

))}

        </tbody>

    </table>

</div>

</div>

) : (

<LoadingAnimation />

)}

</div> : <Accessdenied/>}</>

);
```

PARK BOOK

```javascript
};


export default Adminhist;
```

Update Price Page (updateprice.js):

```javascript
import React, { useEffect, useState } from 'react';


const UpdatePrices = () => {
  const [carPrice, setCarPrice] = useState('');
  const [bikePrice, setBikePrice] = useState('');
  const [fetchedcar, setFetchedcar] = useState('');
  const [fetchedbike, setFetchedbike] = useState('');
  const [message, setMessage] = useState('');
  const [error, setError] = useState('');
  const [count,setcount]=useState(0);



useEffect(() => {





  const fetchPrices = async () => {
    try {
      const response = await fetch('https://park-book-
9f9254d7f86a.herokuapp.com/api/prices');
      const data = await response.json();
      if (response.ok) {
        setFetchedcar(data[0].carprice);
        setFetchedbike(data[0].bikeprice);
```

```
      }

    } catch (error) {

      console.error('Error fetching prices:', error);

    }

  };


  fetchPrices();

}, [count]);


  const handleSubmit = async (event) => {

    event.preventDefault();

    if (isNaN(carPrice) || isNaN(bikePrice) || carPrice <= 0 || bikePrice <= 0) {

      setError('Please enter valid positive numbers for prices.');

      return;

    }


    try {

      const response = await fetch('https://park-book-
9f9254d7f86a.herokuapp.com/api/updatePrices', {

        method: 'POST',

        headers: { 'Content-Type': 'application/json' },

        body: JSON.stringify({ carPrice: parseFloat(carPrice), bikePrice:
parseFloat(bikePrice) })

      });

      const data = await response.json();

      if (response.ok) {

        setMessage('Prices updated successfully!');

        setError('');

      } else {

        throw new Error(data.message || 'Error updating prices');

      }

    } catch (error) {
```

```
      setError(error.message);

      setMessage('');

    }

  };



  return (

      <div className="max-w-md mx-auto mt--10 px-4 py-8 bg-secondary rounded-lg
shadow-xl shadow-gray-500">

        <h1 className="text-2xl text-center text-white font-bold mb-4">Update Parking
Prices</h1>

        <button className='bg-cyan-500 text-black p-1 font-bold rounded-md hover:bg-
cyan-600 animate-pulse' onClick={() => {setcount(count+1)}}> Fetch latest prices



        </button>

        <form onSubmit={handleSubmit}>

          <div className="mb-4">

            <label className="block text-white text-sm font-bold mb-2"
htmlFor="carPrice">

              Car Price (per minute)

            </label>

            <span className='text-cyan-500 animate-pulse'>Current Price: <span
className='text-red-400 animate-pulse'>{fetchedcar}</span> </span>

            <br></br>

            <input

            className="shadow appearance-none rounded w-full py-2 px-3 text-white
leading-tight focus:outline-none focus:shadow-outline bg-primary border border-gray-500"

              id="carPrice"

              type="number"

              placeholder="Enter car price"

              value={carPrice}

              onChange={(e) => setCarPrice(e.target.value)}
```

```
            formNoValidate

        />

    </div>

    <div className="mb-4">

        <label className="block text-white text-sm font-bold mb-2"
htmlFor="bikePrice">

            Bike Price (per minute)

        </label>

        <span className='text-cyan-400 animate-pulse'>Current Price:  <span
className='text-red-400 animate-pulse'>{fetchedbike}</span> </span>

        <br></br>


        <input

            className="shadow appearance-none rounded w-full py-2 px-3 text-white
leading-tight focus:outline-none focus:shadow-outline bg-primary border border-gray-500"

            id="bikePrice"

            type="number"

            placeholder="Enter bike price"

            value={bikePrice}

            onChange={(e) => setBikePrice(e.target.value)}


        />

    </div>

    <div className="flex items-center justify-between">

        <button

            className="bg-white hover:bg-black text-black hover:text-white border
border-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"

            type="submit">

            Update Prices

        </button>

    </div>

    {message && <p className="mt-4 text-green-600">{message}</p>}

    {error && <p className="mt-4 text-red-600">{error}{<span className='text-900
p-1 rounded bg-white '   onClick={()=>{setError("")}}>X</span>}</p>}
```

Govinda Dasa College, Surathkal

```
        </form>

    </div>

  );

};


export default UpdatePrices;
```

Main Page (main.js):

```
import React from 'react';

import { useNavigate } from 'react-router-dom';

import { useAuth0 } from '@auth0/auth0-react';

import Profile from './Profile';

import About from './about';

import './main.css';


const Main = () => {

  const navigate = useNavigate();

  const { loginWithRedirect, logout, isAuthenticated } = useAuth0();


  return (

    <div className="h-screen">

      <div className="flex flex-col h-screen bg-primary">

        <nav className="text-white py-4 px-8 flex justify-between items-center border-b
border-gray-200">

          <h1 className="text-4xl font-bold text-white fam">ParkWay</h1>

          {isAuthenticated ? (

            <div className="flex items-center">

              <Profile />

              <button

                className="bg-white hover:bg-black hover:text-white border-2 border-white text-
black font-bold py-2 px-4 rounded"

                onClick={() => logout({ returnTo: window.location.origin })}
```

```
        >

          Log Out

        </button>

      </div>

    ) : (

      <button

        className="bg-gray-800 hover:bg-gray-700 border border-white hover:bg-white
hover:text-black text-white font-bold py-2 px-4 rounded"

        onClick={() => loginWithRedirect()}

      >

        Get Started

      </button>

    )}

  </nav>



  <div className="flex-grow relative">

   <div className="absolute inset-0" />

   <div className="absolute inset-0 opacity-50" />



   <div className="absolute inset-0 flex flex-col justify-center items-center">

     <p className="text-blue-500 text-4xl sm:text-5xl md:text-7xl font-bold text-center
mb-2 ">

       <span className="block sm:inline text-white">Don't waste your time
anymore</span>

       <br className="sm:hidden" />

       <span className="block sm:inline text-white "> to find a parking space.</span>

     </p>

     <p className="text-slate-300 text-center text-lg sm:text-xl font-semibold">

       Welcome to <span className="name font-bold">ParkWay</span> - your ultimate
solution for booking parking spots online. Book your spot with

       <br className="hidden sm:block" />

       just one click! Our system offers a user-friendly interface for booking, accessing

       <br className="hidden sm:block" />
```

```
      order history, and more.

    </p>

    {!isAuthenticated && (

     <button

      className="bg-white hover:bg-black hover:text-white border border-white text-
black font-bold py-2 px-4 rounded-full mt-4"

      onClick={() => loginWithRedirect()}

     >

      Get Started

     </button>

    )}

    {isAuthenticated && (

     <button

      className="btn bg-blue-500 hover:bg-blue-600 text-white font-bold py-2 px-6
rounded mt-4"

      onClick={() => navigate('/home')}

     >

      Book

     </button>

    )}

    </div>

   </div>

   </div>

   <About />

  </div>

 );

};


export default Main;
```

Home Page (home.js):

```
import React, { useState,useEffect } from 'react';
```

PARK BOOK

```jsx
import { useNavigate } from 'react-router-dom';

import { useAuth0 } from '@auth0/auth0-react';

import { withAuthenticationRequired } from '@auth0/auth0-react';

import Profile from './Profile';

import { FaRegArrowAltCircleRight, FaBars, FaTimes } from "react-icons/fa";

import { GiFallingStar } from "react-icons/gi";

import { IoCarSportSharp } from "react-icons/io5";

import Book from './book';

const HomePage = () => {

  const navigate = useNavigate();

  const { logout } = useAuth0();

  const [isNavOpen, setIsNavOpen] = useState(false);

  const user = useAuth0();

  const [message, setMessage] = useState('');

  const [verified, setVerified] = useState(false);



  useEffect(() => {


   async function checkverify(){

      try{  if(user.user.email){

      const response =await fetch('https://park-book-9f9254d7f86a.herokuapp.com/api/isverified', {

        method: 'POST',

        headers: { 'Content-Type': 'application/json' },

        body: JSON.stringify({ email: user.user.email })

      });

      const data = await response.json();

      console.log(data);

      if (data.isverified===true) {


        setVerified(true);
```

PARK BOOK

```
      }
      else {

        setVerified(false);

      }

      }}catch(error){
        console.error('Failed to verify:', error);
      }




  } checkverify();}
      , [user.user.email]);
  const handleNavigation = (path) => {
   navigate(path);
   setIsNavOpen(false);
 };
function handlebook(e){
 e.preventDefault();
 if(verified){
   navigate('/book');
 }


 else
 setMessage('Please verify your phone number to book a slot');
}



 return (
```

Govinda Dasa College, Surathkal

PARK BOOK

```jsx
  <div className="min-h-screen py-6 flex flex-col justify-center sm:py-12 bg-gray-900 text-white">

  <Profile />


  {/* Navigation bar */}

  <div className="px-4 py-2 shadow-lg bg-gray-800 flex items-center justify-between">

    <img src="https://api.time.com/wp-content/uploads/2015/02/apple-logo.jpg" alt="Logo" className="h-8 w-auto" />


    <div className="md:hidden">

     <button onClick={() => setIsNavOpen(!isNavOpen)} className="text-gray-300">

      <FaBars size={24} />

     </button>

    </div>


    <div className="hidden md:flex space-x-4">

     {[ 'admin',  'history','verify'].map((item) => (

      <button onClick={() => handleNavigation(`/${item}`)} className="bg-gray-700 hover:bg-gray-600 text-gray-200 font-semibold py-2 px-4 rounded">

       {item.charAt(0).toUpperCase() + item.slice(1)}

      </button>

     ))}

     <button onClick={() => logout({ returnTo: window.location.origin })} className="bg-red-600 hover:bg-red-700 text-white font--bold py-2 px-4 rounded">

      Log Out

     </button>


    </div>

   </div>


  {isNavOpen && (

   <div className="fixed inset-0 z--50 flex bg-black bg-opacity-75">

    <div className="flex-1" onClick={() => setIsNavOpen(false)} />
```

Govinda Dasa College, Surathkal

PARK BOOK

```
    <div className="w-64 bg-gray-800 p-4">

    <button onClick={() => setIsNavOpen(false)} className="text-gray-300">

     <FaTimes size={24} />

    </button>

    {[ 'admin', 'history','verify'].map((item) => (

     <button onClick={() => handleNavigation(`/${item}`)} className="block mt-4 text-
gray-200 font-bold py-2 px-4 hover:bg-gray-700 rounded">

       {item.charAt(0).toUpperCase() + item.slice(1)}

     </button>

    ))}

     <button onClick={() => logout({ returnTo: window.location.origin })} className="mt-4
text-gray-200 font-bold py-2 px-4 hover:bg-red-600 rounded">

      Log Out

     </button>

    </div>

   </div>

  )}


    {/* Main content */}
    <div className="relative py-3 sm:max-w-xl sm:mx-auto px-4">

     <div className="absolute inset-0 bg-gradient-to-r from-gray-700 to-gray-900 shadow-
lg transform -skew-y-6 sm:skew-y-0 sm:-rotate-6 rounded-3xl"></div>

     <div className="bg-gray-800 border border-gray-700 relative px-4 py--10 text-white
shadow-lg rounded-3xl sm:p-20">

      <div className="max-w-md mx-auto">

      <div>

       <IoCarSportSharp size={50} color='cyan' className='mx-auto' />

      </div>

      <div className="divide-y divide-gray-700">

       <div className="py-8 text-base leading-6 space-y-4 text-gray-300 sm:text-lg
sm:leading-7">

        <p className='font-bold flex items-center justify-center'>An advanced online
parking booking system <GiFallingStar size={30} color='cyan' className='ml-2' /></p>

        <form className="mt-6">
```

Govinda Dasa College, Surathkal

PARK BOOK

```jsx
        <div>

          <label htmlFor="location" className="sr-only">Location</label>

          <select className='bg-gray-700 rounded text-gray--200'>

            <option value="location">Railway MlR CNTRL</option>

            <option value="location">none</option>

          </select>

        </div>

        <div className="mt-6">

          <button  onClick={handlebook} className="w-full mr-auto px-4 py-2 font-bold
bg-indigo-600 hover:bg-indigo-700 rounded-md focus:outline-none flex justify-center items-
center">Book a parking space <FaRegArrowAltCircleRight size={25} className='ml-1'
/></button>


        </div>

        {message && <p className="mt-4 text-red-600">{message}</p>}

      </form>

     </div>

    </div>

   </div>

  </div>

 );
};


export default withAuthenticationRequired(HomePage, {

  onRedirecting: () => (<div>Loading...</div>)

});
```

Booking Page (book.js):

```jsx
import React, { useState, useEffect } from 'react';

import LoadingAnimation from './steering';

import { useNavigate } from 'react-router-dom';

import { useAuth0 } from "@auth0/auth0-react";
```

Govinda Dasa College, Surathkal

PARK BOOK

```
import Alert from '@mui/material/Alert';

import { BsTicketPerforated } from 'react-icons/bs';

import { IoCarSportSharp } from "react-icons/io5";

import { PiMotorcycleFill } from "react-icons/pi";

import { FaRegArrowAltCircleRight } from "react-icons/fa";

import { MdCancel } from "react-icons/md";

import { GiConfirmed } from "react-icons/gi";


import car from '../images/sport-car.png'

import bike from '../images/bike.webp'


import ConfirmationPage from './confirm';


const Book = () => {
  const navigate = useNavigate();

  const { user } = useAuth0();

  const [isConfirmModalVisible, setIsConfirmModalVisible] = useState(false);

  const [booking, setBooking] = useState({

    vehicleType: "",

    date: "",

    date2: "",

    time: "",

    endTime: "",

    slotId: null,

    vehicleNumber: "",

    amount: 0

  });

  const [slots, setSlots] = useState([]);

  const [bookedFrom, setBookedFrom] = useState("");

  const [bookedTill, setBookedTill] = useState("");

  const [loading, setLoading] = useState(false);

  const [amount, setAmount] = useState("");
```

PARK BOOK

```
const [error, setError] = useState(null);

const [carprice, setCarprice] = useState(0);

const [bikeprice, setBikeprice] = useState(0);

const [confirm, setConfirm] = useState(false);

const [currentPage, setCurrentPage] = useState(1);

const [verified, setVerified] = useState(true);

const slotsPerPage = 16;

const [otp, setotp] = useState('');


useEffect(() => {
  console.log("loading", loading);
}, [loading]);


useEffect(() => {
  const fetchPrice = async () => {
    try {
      const response = await fetch('https://park-book-
9f9254d7f86a.herokuapp.com/api/prices');

      const data = await response.json();

      setCarprice(data[0].carprice);

      setBikeprice(data[0].bikeprice);

      console.log(carprice);

      console.log(bikeprice);

    } catch (error) {

      console.error('Failed to fetch prices:', error);

    }
  };

  fetchPrice();
}, [bookedFrom, bookedTill, booking.vehicleType, carprice, bikeprice]);


useEffect(() => {
  if (bookedFrom && bookedTill && booking.vehicleType && carprice || bikeprice) {
```

```
    const calculatedAmount = calculateAmount(bookedFrom, bookedTill,
booking.vehicleType);

    setAmount(calculatedAmount);

  }
 // eslint-disable-next-line react-hooks/exhaustive-deps
 }, [bikeprice, bookedFrom, bookedTill, booking.vehicleType, carprice]);


 useEffect(() => {
  if (error) {
    setSlots([]);
  } else {
    setBooking(prev => ({ ...prev, vehicleType: !booking.vehicleType }));
  }
 }, [error]);


  async function checkverify() {
   try {
    if (user.user.email) {
      const response = await fetch('https://park-book-
9f9254d7f86a.herokuapp.com/api/isverified', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email: user.user.email })
      });
      const data = await response.json();
      console.log(data);
      if (data.isverified === true) {
       alert('Your phone number is verified');
       setVerified(true);
      } else {
       setVerified(false);
      }
     }
```

Govinda Dasa College, Surathkal

```
    } catch (error) {

      console.error(error);

     }

   }




  useEffect(() => {

    console.log('fetching slots');

    console.log(bookedFrom, bookedTill);

    console.log(booking.vehicleType);


    const fetchSlots = async () => {

      if (!bookedFrom || !bookedTill || !booking.vehicleType || !carprice || !bikeprice ||
!booking.vehicleNumber) {

        return;

      }


      try {

        const url = `https://park-book-
9f9254d7f86a.herokuapp.com/api/parkingSlots?bookedFrom=${bookedFrom}&bookedTill=
${bookedTill}&type=${booking.vehicleType}`;

        const response = await fetch(url);

        const data = await response.json();


        if (Array.isArray(data)) {

          const uniqueSlots = data.filter((slot, index, self) => self.findIndex(s => s.slotId ===
slot.slotId) === index);

          setSlots(uniqueSlots);


        }

      } catch (error) {

        console.error('Failed to fetch slots:', error);
```

Govinda Dasa College, Surathkal

```
     setSlots([]);

     setError("Error fetching slots");

    }

   };


   fetchSlots();

  }, [bookedFrom, bookedTill, booking.vehicleType,
booking.vehicleNumber,bikeprice,carprice]);


 const handleBooking = (e) => {

  e.preventDefault();

  if (!booking.vehicleType || !booking.date || !booking.time || !booking.date2 ||
!booking.endTime || !booking.slotId || !booking.vehicleNumber || !amount) {

    setError('Please fill all the fields');

    return;

  }



   setIsConfirmModalVisible(true);

 };

 const mask = 'AA-11-AA-1111';

 const regex = {

  A: /[A-Z]/,

  1: /\d/,

  '-': /-/

 };

 const handleChangee = (event) => {

  const { name, value } = event.target;


  if (name === "vehicleNumber") {

   let idx = 0, valid = true;

   let newValue = '';

   const splitValue = value.split('');
```

Govinda Dasa College, Surathkal

```
    for (const char of splitValue) {
     if (idx >= mask.length || !regex[mask[idx]].test(char)) {
      valid = false;
      break;
     } else {
      newValue += char;
      idx++;
     }
    }


    if (!valid) {
     setError("Invalid vehicle number format. Expected format: KA-19-HC-0123");
     return; // Prevent further state update if format is incorrect
    } else {
     setError(null); // Clear error if format is correct
    }


    setBooking(prev => ({
     ...prev,
     [name]: newValue.toUpperCase() // Ensuring input is uppercase
    }));
   } else {
    // Handle other inputs normally
    setBooking(prev => ({
     ...prev,
     [name]: value
    }));
   }
  };
 const confirmBooking = async () => {
  setLoading(true);
```

Govinda Dasa College, Surathkal

```
    const response = await fetch('https://park-book-
9f9254d7f86a.herokuapp.com/api/bookings', {

    method: 'POST',

    headers: { 'Content-Type': 'application/json' },

    body: JSON.stringify({

      userId: user.email,

      vehicleType: booking.vehicleType,

      amount: amount,

      bookedFrom: `${booking.date} ${booking.time}`,

      bookedTill: `${booking.date2} ${booking.endTime}`,

      slotId: booking.slotId,

      vehicleNumber: booking.vehicleNumber,

    }),

  });


    const data = await response.json();

    if (response.ok) {

     setLoading(false);

     console.log('Booking successful:', data);


     setConfirm(true);

    } else {

     setLoading(false);

     setError(data.message);

     console.error('Failed to book:', data);

    }

  };


  const calculateAmount = (from, till, type) => {

   const start = new Date(from);

   const end = new Date(till);

   const diff = (end - start) / 60000; // difference in minutes
```

PARK BOOK

```javascript
    const rate = type === "car" ? carprice : bikeprice;
    const calculatedAmount = diff * rate;
    try {
     if (calculatedAmount <= 0) {
      setError('Calculated amount is non-positive');
      throw new Error('Calculated amount is non-positive');
     }
     if (calculatedAmount < 10 && !(calculatedAmount > 10)) {
      setError('Minimum amount is 10');
      return 10;
     }
     return calculatedAmount;
    } catch (error) {
     console.error('Error calculating amount:', error);
    }
   };


  const handleChange = (event) => {
   event.preventDefault();
    checkverify();
   if (!verified) {
    setError('Please  your phone number to book a slot');
    navigate('/verify');
    return;
   }
   const { name, value } = event.target;
   setBooking(prev => ({ ...prev, [name]: value }));


   const { date, time, date2, endTime } = { ...booking, [name]: value };
   const now = new Date();


   if (date && time && date2 && endTime) {
```

PARK BOOK

```javascript
    const startDateTime = new Date(`${date}T${time}`);

    const endDateTime = new Date(`${date2}T${endTime}`);


    if (startDateTime < now || endDateTime < now) {

      setError('Booking times must be in the future');

      setBookedFrom("");

      setBookedTill("");

    } else if (startDateTime >= endDateTime) {

      setError('End time should be later than start time');

      setBookedFrom("");

      setBookedTill("");

    } else {

      setBookedFrom(`${date}T${time}`);

      setBookedTill(`${date2}T${endTime}`);

      setError(null);

    }

  }

  if (name === "vehicleType") {

    setCurrentPage(1); // Reset to the first page when vehicle type changes

  }

};


const handleSlotSelect = (slotId) => {

  const slot = slots.find(s => s.slotId === slotId);

  if (slot.isOccupied) {

    setError('Slot is already occupied');

  } else {

    setBooking(prev => ({ ...prev, slotId }));

  }

};


const handleNextPage = (event) => {
```

95

```
    event.preventDefault();

    setCurrentPage(prevPage => prevPage + 1);

  };


  const handlePreviousPage = (event) => {

   event.preventDefault();

   setCurrentPage(prevPage => prevPage - 1);

  };


  const currentSlots = slots.slice((currentPage - 1) * slotsPerPage, currentPage *
slotsPerPage);


//delay in selecting vehicleType
  const handleVehicleTypeChange = (vehicleType) => {

   setLoading(true);

   setBooking((prev) => ({

     ...prev,

     vehicleType,

   }));

   setCurrentPage(1);

   setLoading(false);

  };



  return (

   <>

    {!confirm ?

    <div className="bg-primary min-h-screen bg-neutral-900 text-white py-4 px-4 sm:px-2
md:px-6 lg:px-10">

     <div className="container mx-auto py-8">

      <div className="bg-neutral-800 p-4 sm:p-6 lg:p-8 rounded-2xl shadow-xl shadow-gray-
500">

       <h1 className="text-2xl sm:text-3xl font-bold mb-6">Parking Booking</h1>
```

96

Govinda Dasa College, Surathkal

PARK BOOK

```
<form id="bookingform">

  <div className="mb-4">

    <label htmlFor="vehicleType" className="font-medium">Vehicle Type:</label>

    <div className="flex gap-4 mt-2">

      <button

        type="button"

        onClick={() => handleVehicleTypeChange("car")}

        className={`py-2 px-4 rounded-md border border-gray-400 transition-colors
${booking.vehicleType === "car" ? "bg-cyan-500" : "bg-primary hover:bg-gray-600"} font-
bold`}

      >

        <IoCarSportSharp size={30} className="text-black-500" />

        Car

      </button>

      <button

        type="button"

        onClick={() => handleVehicleTypeChange("bike")}

        className={`py-2 px-4 rounded-md border border-gray-400 transition-colors
${booking.vehicleType === "bike" ? "bg-cyan-500" : "bg-primary hover:bg-gray-600"} font-
bold`}

      >

        <PiMotorcycleFill size={30} className="text-black-500" />

        Bike

      </button>

    </div>

  </div>

  <div className="grid grid-cols-1 md:grid-cols-2 gap-4 mb-4">

    <div>

      <label htmlFor="date" className="font-medium">Start Date:</label>

      <input type="date" name="date" required className="form-input w-full mt-1 p-2
bg-gray-700 border border-gray-400 rounded bg-primary picker" onChange={handleChange}
value={booking.date} />

    </div>

    <div>
```

Govinda Dasa College, Surathkal

```jsx
        <label htmlFor="time" className="font-medium">Start Time:</label>

        <input type="time" name="time" required className="form-input w-full mt-1 p-2
bg-gray-700 border border-gray-400 rounded bg-primary picker" onChange={handleChange}
value={booking.time} />

        </div>

        <div>

        <label htmlFor="date2" className="font-medium">End Date:</label>

        <input type="date" name="date2" required className="form-input w-full mt-1 p-2
bg-gray-700 border border-gray-400 rounded bg-primary picker" onChange={handleChange}
value={booking.date2} />

        </div>

        <div>

        <label htmlFor="endTime" className="font-medium">End Time:</label>

        <input type="time" name="endTime" required className="form-input w-full mt-1
p-2 bg-gray-700 border border-gray-400 rounded bg-primary picker"
onChange={handleChange} value={booking.endTime} />

        </div>

        </div>

        <div>

    <label htmlFor="vehicleNumber" className="font-medium">Vehicle Number:</label>

    <input

      type="text"

      name="vehicleNumber"

      placeholder="KA-19-HC-0123"

      onChange={handleChangee}

      required

      className="form-input w-full mt-1 p-2 bg-gray-700 border border-gray-400 rounded bg-
primary"

        // Use the specific change handler for vehicle number

      value={booking.vehicleNumber}

    />

    {error && <div className="bg-red-700 text-center p-3 rounded mb-4">{error}</div>}

   </div>

        {error && <div className="bg-red-700 text-center p-3 rounded mb-4">

        {error}
```

Govinda Dasa College, Surathkal

PARK BOOK

```
    </div>}

    {/* Total Charges Display */}

    {amount > 0 ? <p>Total Charges:<span className='font-medium text-cyan-500'>INR:
{amount.toFixed(2)}</span> </p> : <span className="relative flex h-3 w-3">Fetching

      <span className="animate-ping absolute inline-flex h-full w-full rounded-full bg-
sky-400 opacity-75"></span>

      <span className="relative inline-flex rounded-full h-3 w-3 bg-sky-500"></span>

    </span>}

    {/* Slot Selection */}

    <div className="mb-6 mt-2">

      <label className="block text-sm font-medium text-gray-700">Select Slot</label>

      {!loading ? (

        <div className="flex justify-center">

        <div className=" grid grid-cols-2 sm:grid-cols-2 md:grid-cols-8 gap-4 mt-2 ">

          {currentSlots.map(slot => (

            <div key={slot.slotId} onClick={() => handleSlotSelect(slot.slotId)}

              className={`flex flex-col justify-center text-center items-center cursor-pointer
p-4 w-24 h-22 sm:w-32 sm:h-32 md:w-32 md:h-32 ${slot.slotId === booking.slotId ? 'bg-
cyan-400 text-black' : slot.isOccupied ? 'bg-slate-500 text-white' : 'bg-slate-500 text-white'}
rounded-lg`}>




              {slot.isOccupied ? booking.vehicleType === "car" ? <img src={car} alt="Car"
width="100" height="50"/> : <img src={bike} className='' alt="Bike" width="100"
height="50"/> :null}




              {slot.isOccupied? <p>Slot {slot.slotId} </p> : <p>Slot {slot.slotId}
<br/>Available</p>}




            </div>

          ))}

        </div></div>

      ) : (

        <LoadingAnimation />
```

99

```
            )}

            {slots.length > 0 && (

              <div className="flex justify-between mt-4">

                <button disabled={currentPage === 1} onClick={handlePreviousPage}
className="bg-gray-400 text-black font-bold py-2 px-4 rounded disabled:opacity-50">

                  Previous

                </button>

                <button disabled={currentSlots.length < slotsPerPage} onClick={handleNextPage}
className="bg-gray-400 text-black font-bold py-2 px-4 rounded disabled:opacity-50">

                  Next

                </button>

              </div>

            )}

            {error && <Alert variant="filled" severity="error" onClose={() => { setError(null)
}}>{error}</Alert>}

          </div>

          {/* Booking Button */}

          <button type="submit" className="w-full bg-white text-black font-bold py-2 px-4
rounded-md hover:bg-black hover:text-white border border-gray-400 flex justify-center
items-center" onClick={handleBooking}>

            Book Now <FaRegArrowAltCircleRight size={25} className='ml-1' />

          </button>

        </form>

      </div>

    </div>

    {/* Confirmation Modal */}

    {isConfirmModalVisible && (

      <div className="fixed inset-0 bg-black bg-opacity-30 backdrop-blur-sm flex items-
center justify-center p-4">

        <div className="bg-slate-700 rounded-lg p-6 max-w-sm mx-auto">

          <h2 className="text-lg font-semibold">Confirm Booking</h2>

          <p className="text-sm">Please confirm your booking details:</p>

          <ul className="text-sm list-disc pl-5 mt-2">

            <li>Vehicle Type: {booking.vehicleType}</li>
```

```
        <li>Vehicle Number: {booking.vehicleNumber}</li>

        <li>Start Date: {booking.date}</li>

        <li>Start Time: {booking.time}</li>

        <li>End Date: {booking.date2}</li>

        <li>End Time: {booking.endTime}</li>

        <li>Slot: {booking.slotId}</li>

        <li>Amount: INR: {amount.toFixed(2)}</li>

      </ul>

      <div className="flex justify-end gap-4 mt-4">

      <button onClick={() => setIsConfirmModalVisible(false)} className="bg-red-300
hover:bg-gray-400 text-black font-bold py-2 px-4 rounded flex justify-center items-center">

        Cancel<MdCancel className="ml-2" />

      </button>

      {!loading ? (

        <button onClick={confirmBooking} className="bg-cyan-500 hover:bg-sky-600 text-
black font-bold py-2 px-4 rounded flex justify-center items-center">

         Pay and Confirm<GiConfirmed className="ml-2" />

        </button>

      ) : (

        <button className="bg-blue-500 hover:bg-blue-600 text-white py-2 px-4
rounded">

          <BsTicketPerforated className="animate-spin" />

        </button>

      )}

      </div>

     </div>

    </div>

   )}

  </div>

    :

  <ConfirmationPage code={otp} />}

  </>

);
```

Govinda Dasa College, Surathkal

PARK BOOK

```js
};


export default Book;
```

Confirm Page (confirm.js):

```js
import React from 'react';

import { useNavigate } from 'react-router-dom';

import './confirm.css';


function ConfirmationPage() {
  const navigate = useNavigate();


  return (

    <div className="min-h-screen bg-neutral-900 text-white bg-primary flex items-center justify-center">

      <div className="w-full max-w-lg mx-auto p-4 sm:p-8">

        <div className="p-6 bg-neutral-800 rounded-lg shadow-lg border-2 border-gray-400">

          <div className="flex justify-center mb-0">

            <svg className="funds-checkmark" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 52 52"><circle className="funds-checkmark-circle" cx="26" cy="26" r="25" fill="none" /><path className="funds-checkmark-check" fill="none" d="M14.1 27.2l7.1 7.2 16.7-16.8" /></svg>

          </div>

          <audio src="success.mp3" autoPlay></audio>

          <div className="text-center">

            <h1 className="text-white text-3xl font-bold mb-2 delayed-text">Done!</h1>

            <h2 className="text-white text-xl mb-4 delayed-text">Your Booking has been confirmed!</h2>

            <p className="text-white text-lg delayed-text">Check your otp in history</p>

            <div className="flex flex-col sm:flex-row justify-center space-y-2 sm:space-y-0 sm:space-x-4">

              <button

                className="bg-white border border-white text-neutral-900 font-bold py-2 px-4 rounded-md hover:bg-black hover:text-white delayed-text"
```

```
        onClick={() => navigate('/home')}

      >

       Go to Home

     </button>

     <button

        className="bg-white border border-white text-neutral-900 font-bold py-2 px-4
rounded-md hover:bg-black hover:text-white delayed-text"

        onClick={() => navigate('/history')}

      >

       View History

     </button>

    </div>

   </div>

  </div>

 </div>

 );

}


export default ConfirmationPage;
```

History Page (history.js):

```
import React, { useEffect, useState } from 'react';

import { useAuth0 } from '@auth0/auth0-react';

import Alert from '@mui/material/Alert';

import LoadingAnimation from './steering';


const History = () => {

  const { user } = useAuth0();

  const [bookings, setBookings] = useState([]);

  const [count, setCount] = useState(0);

  const [loading, setLoading] = useState(false);
```

PARK BOOK

```javascript
useEffect(() => {

  const fetchUserBookings = async (userEmail) => {

    try {

      const url = `https://park-book-
9f9254d7f86a.herokuapp.com/api/userBookings?email=${encodeURIComponent(userEmail
)}`;

      const response = await fetch(url);

      if (response.ok) {

        const bookings = await response.json();

        console.log('User bookings:', bookings);

        // Sort bookings by date

        bookings.sort((a, b) => new Date(b.bookedFrom) - new Date(a.bookedFrom));

        setBookings(bookings);

      } else {

        throw new Error('Failed to fetch bookings');

      }

    } catch (error) {

      console.error('Error fetching user bookings:', error);

    }

  };

  if (user) {

    fetchUserBookings(user.email);

    setLoading(false);

  } else {

    setLoading(true);

  }

}, [user, count]);


const isBookingExpired = (bookedTill) => {

  const now = new Date();

  const tillDate = new Date(bookedTill);

  return now > tillDate;

};
```

PARK BOOK

```
    return (
      <div className="bg-primary text-white min-h-screen ">
        <div className="flex justify-between items-center text-2xl font-bold mb-4 px-2 py-4
border-b border-gray-500 bg-secondary  ">
          <h1 className="l-border fam ">Booking History</h1>
          <button onClick={() => setCount(count + 1)} className="bg-white hover:bg-black
hover:text-white text-black font-bold py-1 px-2 border border-white ml-2 rounded-md">
            Refresh
          </button>
        </div>


        {!loading ? (
          <div className="px-4 py-4">
            <div className="overflow-x-auto">
              <table className="min-w-full border rounded-lg fam">
                <thead>
                  <tr>
                    <th className="border border-gray-300 px-4 py-2">Booking ID</th>
                    <th className="border border-gray-300 px-4 py-2">Vehicleno</th>
                    <th className="border border-gray-300 px-4 py-2">Amount</th>
                    <th className="border border-gray-300 px-4 py-2">Slot ID</th>
                    <th className="border border-gray-300 px-4 py-2">Booked From</th>
                    <th className="border border-gray-300 px-4 py-2">Booked Till</th>
                    <th className="border border-gray-300 px-4 py-2">Checkin OTP</th>
                    <th className="border border-gray-300 px-4 py-2">Checkout OTP</th>
                  </tr>
                </thead>
                <tbody>
                  {bookings.map((booking) => (
                    <tr key={booking.bookingId}>
                      <td className="border px-4 py-2">{booking.bookingId}</td>
                      <td className="border px-4 py-2">{booking.vehicleNumber}</td>
```

```
            <td className="border px-4 py-2">{booking.amount}</td>

            <td className="border px-4 py-2">{booking.slotId}</td>

            <td className="border px-4 py-2">{booking.bookedFrom}</td>

            <td className="border px-4 py-2">{booking.bookedTill}</td>

            <td className="border px-4 py-2 font-bold">

              {!booking.isCheckedIn ?

                (isBookingExpired(booking.bookedTill) ? (

                  <span className="text-red-600 font-bold">Booking
Expired</span>

                ) : (

                  booking.checkinotp

                )) : (

                <p className="text-green-600 font-bold">Checked in</p>

              )}

            </td>

            <td className="border px-4 py-2">

              {!booking.isCheckedOut ? (

                booking.checkoutotp ? (

                  booking.checkoutotp

                ) : (

                  <p className="text-red-600 font-bold">Not Checked In</p>

                )

              ) : (

                <p className="text-green-600 font-bold">Checked Out</p>

              )}

            </td>

          </tr>

        ))}

      </tbody>

    </table>

  </div>

</div>
```

106

```
      ) : (
         <LoadingAnimation />
      )}
    </div>
  );
};


export default History;
```

Govinda Dasa College, Surathkal

INTERFACE

PARK BOOK

8. INTERFACE

Main page:

PARK BOOK

Login/Registration page:

Welcome

Log in to dev-3xz0udbsxxtdl6r4 to continue to My App.

G   Continue with Google

OR

Email address*
deveshpoojary@gmail.com

Password*
••••••••

Forgot password?

Continue

Don't have an account? Sign up

Home page:

Welcome, Deveshpoojary

Admin   History   Verify   Log Out

An advanced online parking booking system

Railway MIR CNTRL

Book a parking space →

Govinda Dasa College, Surathkal

PARK BOOK

Admin page(Update price):

Admin history page:

Add/Remove slot page:

Govinda Dasa College, Surathkal

PARK BOOK

Current slots page:

User Booking page:

Slots:
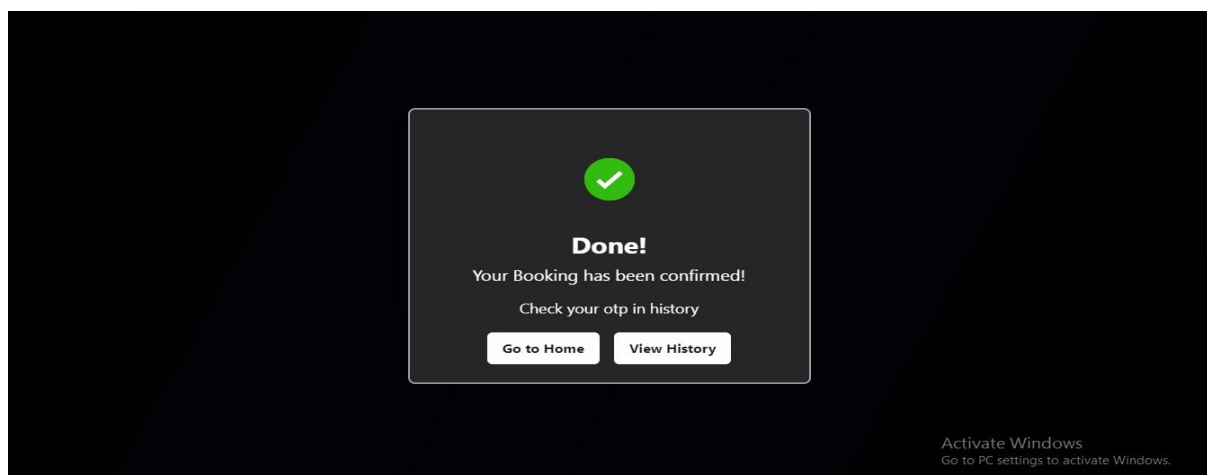
Govinda Dasa College, Surathkal

PARK BOOK

Confirm page:
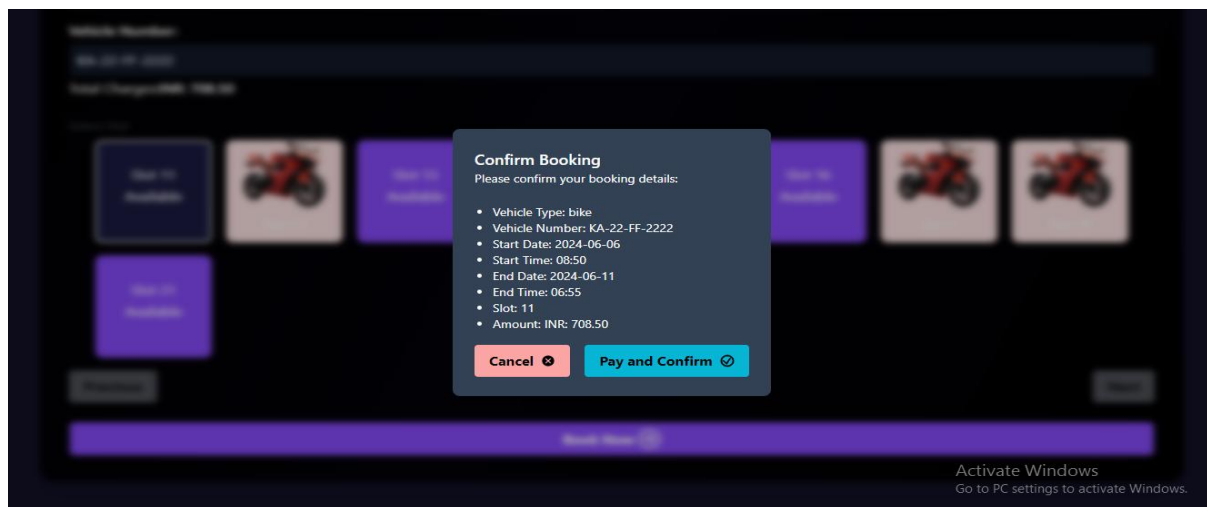
Confirm page:

Booking History page:

Booking ID	Vehicleno	Amount	Slot ID	Booked From	Booked Till	Checkin OTP	Checkout OTP
9	KA-22-FF-2222	708.5	4	2024-06-06 08:50	2024-06-11 06:55	0950	Not Checked In
8	KA-88-HH-7777	4230	20	2024-06-06 06:34	2024-06-12 03:34	9437	Not Checked In
4	KA-11-AA-2222	43436	2	2024-06-05 01:52	2024-06-20 03:50	1100	Not Checked In

CONCLUSION

9. CONCLUSION

The smart PARK BOOK system is a web based application built with React.js and Tailwind for the front-end interface, Node.js and Express for the back-end server, and Mongo DB for database management.

Its purpose is to improve the parking experience for drivers by allowing them to reserve parking spots in advance and providing the real-time information about parking availability in the parking lot.

The project leverages these technologies to create an efficient and user-friendly parking solution. Users can create accounts, log in securely, reserve parking spot.

Admins have access to a dashboard for managing parking lot data and user information. The system aims to reduce the time and effort required to find a parking spot, benefiting both drivers and parking lot operators.

BIBLIOGRAPHY

10. BIBLIOGRAPHY

- https://chat.openai.com
- https://react-icons.github.io
- https://stackoverflow.com